5SSD0

Bayesian Machine Learning and Information Processing

Lecture Notes

Bert de Vries (Flux 7.101)

Wouter Kouw (Flux 7.060)

5SSD0 Course Syllabus

Learning Goals

- This course provides an introduction to Bayesian machine learning and information processing systems. The Bayesian approach affords a unified and consistent treatment of many useful information processing systems.
- Upon successful completion of the course, students should be able to:
  - understand the essence of the Bayesian approach to information processing.
  - specify a solution to an information processing problem as a Bayesian inference task on a probabilistic model.
  - design a probabilistic model by a specifying a likelihood function and prior distribution;
  - Code the solution in a probabilistic programming package.
  - execute the Bayesian inference task either analytically or approximately.
  - evaluate the resulting solution by examination of Bayesian evidence.
  - be aware of the properties of commonly used probability distribitions such as the Gaussian, Gamma and multinomial distribution; models such as hidden Markov models and Gaussian mixture models; and inference methods such as the Laplace approximation, variational Bayes and message passing in a factor graph.

Entrance Requirements (pre-knowledge)

- Undergraduate courses in Linear Algebra and Probability Theory (or Statistics).
- Some scientific programming experience, eg in MATLAB or Python. In this class, we use the Julia programming language, which has a similar syntax to MATLAB, but is (close to) as fast as C.

Important Links

- Please bookmark the following three websites:

  1. The course homepage http://bmlip.nl (or try https://biaslab.github.io/teaching/bmlip) contains links to all materials such as lecture notes and video lectures.
  2. The Piazza course site will be used for Q&A and communication.
  3. The Canvas course site will be sparingly used for communication (mostly by ESA staff)

Materials

- All materials can be accessed from the course homepage.
- Materials consist of the following resources:
  - Mandatory
    - Lecture notes
    - Probabilistic Programming (PP) notes
    - The lecture notes and probabilistic programming notes contain the mandatory materials. Some lecture notes are extended by a reading assignment, see the first cell in the lecture notes. These reading assignment are also part of the mandatory materials.
  - Optional materials to help understand the lecture and PP notes
    - video recordings of the Q2-2023 lecture series
    - exercises
    - Q&A at Piazza
    - practice exams
- Source materials are available at github repo at https://github.com/bertdv/BMLIP. You do not need to bother with this site. If you spot an error in the materials, please raise the issue at Piazza.

Study Guide

- Slides that are not required for the exam are moved to the end of the notes and preceded by an OPTIONAL SLIDES header.
- Please study the lecture notes before you come to class!!
- Optionally, you can view the video recordings of the Q2-2023 lecture series for addional explanations.
- Then come to the class!
  - During the scheduled classroom meetings, I will not teach all materials in the lecture notes.
  - Rather, I will first discuss a summary of the lecture notes and then be available for any additional questions that you may still have.
- Still got any sticky issues regarding the lecture notes?

    - Pose you question at the **Piazza site**!
    - Your questions will be answered at the Piazza site by fellow students and accorded (or corrected) by the teaching staff.
- Each class also comes with a set of exercises. They are often a bit challenging and test more of your quantitative skills than you will need for the exam. When doing exercises, feel free to make use of Sam Roweis' cheat sheets for Matrix identities and Gaussian identities. Also accessible from the course homepage.

Piazza (Q&A)

- We will be using Piazza for Q&A and news. The system is highly catered to getting you help fast and efficiently from both classmates and the teaching staff.
- Sign up for Piazza today if you have not done so. And install the Piazza app on your phone!
- The quicker you begin asking questions on Piazza (rather than via emails), the quicker you'll benefit from the collective knowledge of your classmates and instructors. We encourage you to ask questions when you're struggling to understand a concept—you can even do so anonymously.
- We will also disseminate news and announcements via Piazza.
- Unless it is a personal issue, pose your course-related questions at Piazza (in the right folder).
- Please contribute to the class by answering questions at Piazza.
    - If so desired, you can contribute anonymously.
    - Answering technical questions at Piazza is a great way to learn. If you really want to understand a topic, you should try to explain it to others.
    - Every question has just a single students' answer that students can edit collectively (and a single instructors' answer for instructors).
- You can use LaTeX in Piazza for math (and please do so!).
- Piazza has a great `search` feature. Use search before putting in new questions.

Exam Guide

- The course will be scored by two programming assignments and a final written exam. See the course homepage for how the final score is computed.
- The written exam in multiple-choice format.
- You are not allowed to use books nor bring printed or handwritten formula sheets to the exam. Difficult-to-remember formulas are supplied at the exam sheet.
- No smartphones at the exam.
- The tested material consists of the mandatory lecture + PP notes (+ mandatory reading assignments as assigned in the first cell/slide of each lecture notebook).
- The class homepage contains two representative practice exams from previous terms.

OPTIONAL SLIDES

Title

- The slides below the `OPTIONAL SLIDES` marker are optional for the exam.

Machine Learning Overview

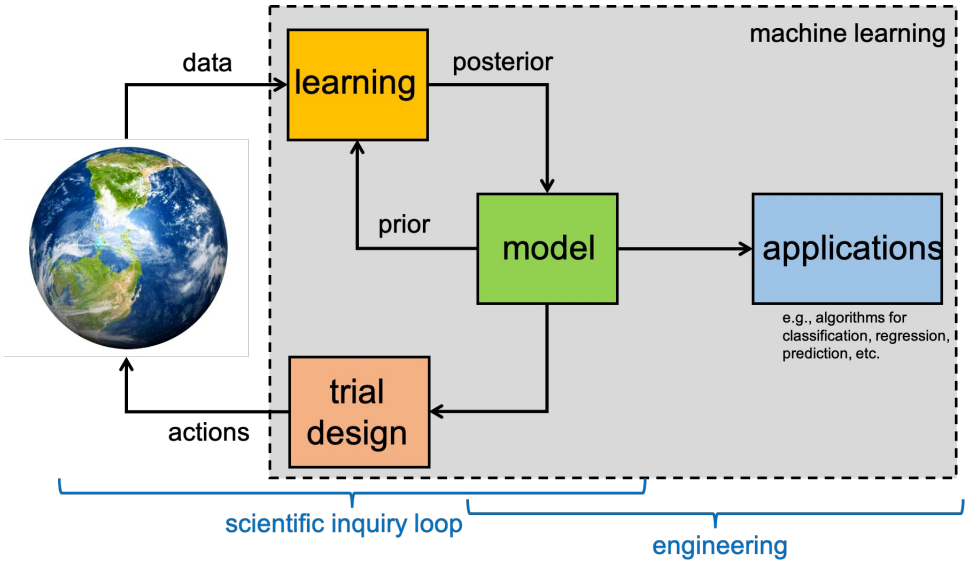
Preliminaries

- Goal
    - Top-level overview of machine learning
- Materials
    - Mandatory
        - this notebook
    - Optional
        - Study Bishop pp. 1-4

What is Machine Learning?

- Machine Learning relates to **building models from data and using these models in applications**.
- **Problem**: Suppose we want to develop an algorithm for a complex process about which we have little knowledge (so hand-programming is not possible).
- **Solution**: Get the computer to develop the algorithm by itself by showing it examples of the behavior that we want.
- Practically, we choose a library of models, and write a program that picks a model and tunes it to fit the data.
- This field is known in various scientific communities with slight variations under different names such as machine learning, statistical inference, system identification, data mining, source coding, data compression, data science, etc.

Machine Learning and the Scientific Inquiry Loop

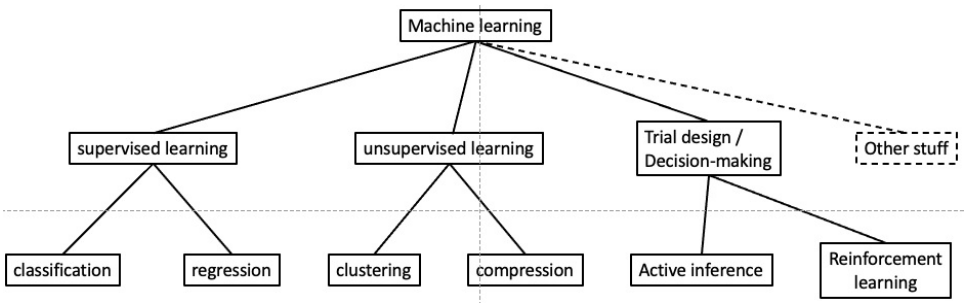


- Machine learning technology uses the scientific inquiry loop to develop models and use these models in applications.

Machine Learning is Difficult

- Modeling (Learning) Problems
  - Is there any regularity in the data anyway?
  - What is our prior knowledge and how to express it mathematically?
  - How to pick the model library?
  - How to tune the models to the data?
  - How to measure the generalization performance?
- Quality of Observed Data
  - Not enough data
  - Too much data?
  - Available data may be messy (measurement noise, missing data points, outliers)

A Machine Learning Taxonomy



- **Supervised Learning**: Given examples of inputs and corresponding desired outputs, predict outputs on future inputs.
  - Examples: classification, regression, time series prediction

- **Unsupervised Learning**: (a.k.a. **density estimation**). Given only inputs, automatically discover representations, features, structure, etc.
  - Examples: clustering, outlier detection, compression

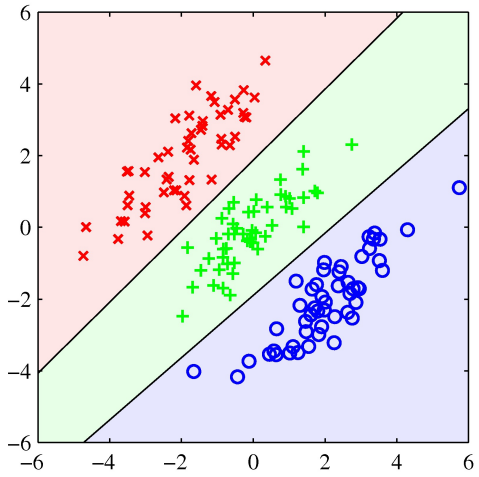
- **Trial Design**: (a.k.a. experimental design, active learning). Learn to make actions that optimize some performance criterion about the expected future.
  - Examples: playing games like chess, self-driving cars, robotics.
  - Two major approaches include **reinforcement learning** and **active inference**
    - **Reinforcement Learning**: Given an observed sequence of input signals and (occasionally observed) rewards for those inputs, *learn* to select actions that maximize *expected* future rewards.
    - **Active inference**: Given an observed sequence of input signals and a prior probability distribution about future observations, *learn* to select actions that minimize *expected* prediction errors (i.e., minimize actual minus predicted sensation).

- Other stuff, like **preference learning**, **learning to rank**, etc., can often be (re-)formulated as special cases of either a supervised, unsupervised or trial design problem.

Supervised Learning

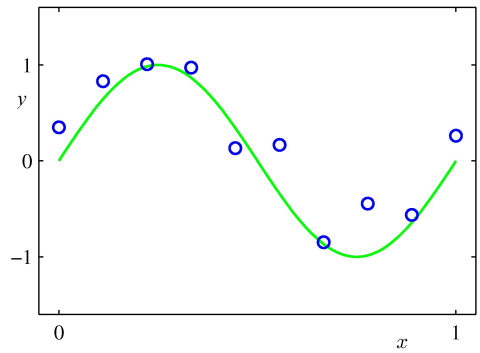
- Given observations of desired input-output behavior $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ (with x_i inputs and y_i outputs), the goal is to estimate the conditional distribution $p(y|x)$ (i.e., how does y depend on x?).

Classification



- The target variable y is a *discrete-valued* vector representing class labels
- The special case $y \in \{\text{true}, \text{false}\}$ is called **detection**.

Regression

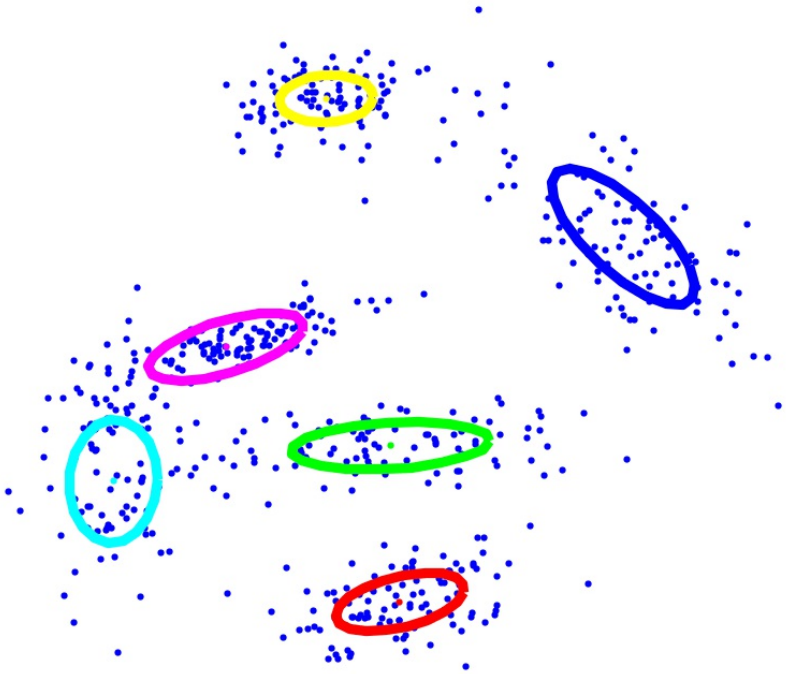


- Same problem statement as classification but now the target variable is a *real-valued* vector.
- Regression is sometimes called **curve fitting**.

Unsupervised Learning

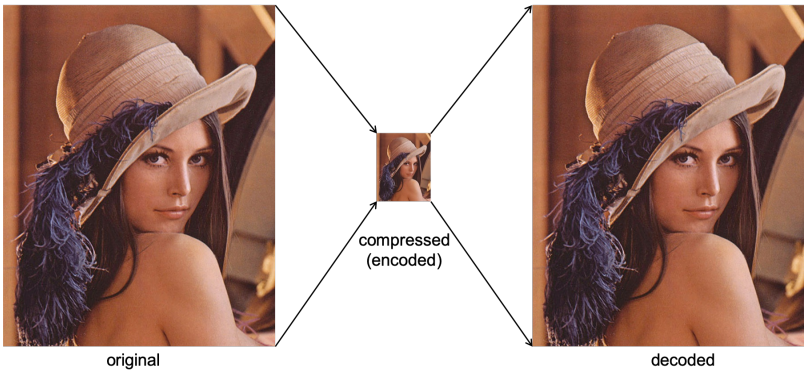
Given data $D = \{x_1, \ldots, x_N\}$, model the (unconditional) probability distribution $p(x)$ (a.k.a. **density estimation**). The two primary applications are **clustering** and **compression** (a.k.a. dimensionality reduction).

Clustering



- Group data into clusters such that all data points in a cluster have similar properties.
- Clustering can be interpreted as "unsupervised classification".

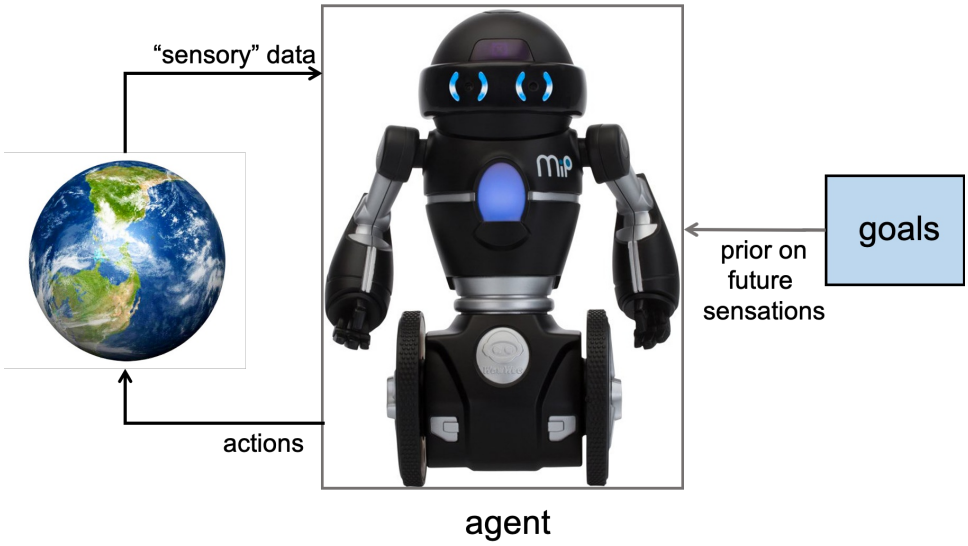
Compression / dimensionality reduction



- Output from coder is much smaller in size than original, but if coded signal if further processed by a decoder, then the result is very close (or exactly equal) to the original.
- Usually, the compressed image comprises continuously valued variables. In that case, compression can be interpreted as "unsupervised regression".

Trial Design and Decision-making

- Given the state of the world (obtained from sensory data), the agent must *learn* to produce actions (like making a movement or making a decision) that optimize some performance criterion about the expected future.



- In contrast to supervised and unsupervised learning, an agent is able to affect its data set by making actions, e.g., a robot can change its input video data stream by turning the head of its camera.
- In this course, we focus on the active inference approach to trial design, see the Intelligent Agent lesson for details.

Some Machine Learning Applications

- computer speech recognition, speaker recognition
- face recognition, iris identification
- printed and handwritten text parsing
- financial prediction, outlier detection (credit-card fraud)
- user preference modeling (amazon); modeling of human perception
- modeling of the web (google)
- machine translation
- medical expert systems for disease diagnosis (e.g., mammogram)
- strategic games (chess, go, backgammon), self-driving cars

- In summary, **any 'knowledge-poor' but 'data-rich' problem**

Probability Theory Review

Preliminaries

- Goal
  - Review of Probability Theory as a theory for rational/logical reasoning with uncertainties (i.e., a Bayesian interpretation)
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 12-24
    - Ariel Caticha, Entropic Inference and the Foundations of Physics (2012), pp.7-56 (ch.2: probability)
      - Great introduction to probability theory, in particular w.r.t. its correct interpretation as a state-of-knowledge.
      - Absolutely worth your time to read the whole chapter, even if you skip section 2.2.4 (pp.15-18) on Cox's proof.
    - Edwin Jaynes, Probability Theory--The Logic of Science (2003).
      - Brilliant book on Bayesian view on probability theory. Just for fun, scan the annotated bibliography and references.
    - Aubrey Clayton, Bernoulli's Fallacy--Statistical Illogic and the Crisis of Modern Science (2021)
      - A very readable account on the history of statistics and probability theory. Discusses why most popular statistics recipes are very poor scientific analysis tools. Use probability theory instead!
    - Joram Soch et al ., The Book of Statistical Proofs (2023 -)
      - On-line resource for proofs in probability theory and statistical inference.

Data Analysis: A Bayesian Tutorial

- The following is an excerpt from the book Data Analysis: A Bayesian Tutorial (2006), by D.S. Sivia with J.S. Skilling:

Preface

As an undergraduate, I always found the subject of statistics to be rather mysterious. This topic wasn't entirely new to me, as we had been taught a little bit about probability earlier at high school; for example, I was already familiar with the binomial, Poisson and normal distributions. Most of this made sense, but only seemed to relate to things like rolling dice, flipping coins, shuffling cards and so on. However, having aspirations of becoming a scientist, what I really wanted to know was how to analyse experimental data. Thus, I eagerly looked forward to the lectures on statistics. Sadly, they were a great disappointment. Although many of the tests and procedures expounded were intuitively reasonable, there was something deeply unsatisfactory about the whole affair: there didn't seem to be any underlying basic principles! Hence, the course on 'probability and statistics' had led to an unfortunate dichotomy: probability made sense, but was just a game; statistics was important, but it was a bewildering collection of tests with little obvious rhyme or reason. While not happy with this situation, I decided to put aside the subject and concentrate on real science. After all, the predicament was just a reflection of my own inadequacies and I'd just have to work at it when the time came to really analyse my data.

The story above is not just my own, but is the all too common experience of many scientists. Fortunately, it doesn't have to be like this. What we were not told in our undergraduate lectures is that there is an alternative approach to the whole subject of data analysis which uses only probability theory. In one sense, it makes the topic of statistics entirely superfluous. In another, it provides the logical justification for many of the prevalent statistical tests and procedures, making explicit the conditions and approximations implicitly assumed in their use.

- Does this fragment resonate with your own experience?
- In this lesson we introduce *Probability Theory* (PT) again. As we will see in the next lessons, PT is all you need to make sense of machine learning, artificial intelligence, statistics, etc.

Challenge: Disease Diagnosis

- **Problem**: Given a disease with prevalence of 1% and a test procedure with sensitivity ('true positive' rate) of 95% and specificity ('true negative' rate) of 85% , what is the chance that somebody who tests positive actually has the disease?
- **Solution**: Use probabilistic inference, to be discussed in this lecture.

The Design of Probability Theory

- Define an **event** (or "proposition") A as a statement that can be considered for its truth by a person. For instance,

$$A = \texttt{'there is life on Mars'}$$

- If we assume the fact

$$I = \texttt{'All known life forms require water'}$$

as background information, and a new piece of information

$$x = \texttt{'There is water on Mars'}$$

becomes available, how *should* our degree of belief in event A be affected *if we were rational*?

- Richard T. Cox (1946) developed a **calculus for rational reasoning** about how to represent and update the degree of *beliefs* about the truth value of events when faced with new information.
- In developing this calculus, only some very agreeable assumptions were made, including:
  - (Representation). Degrees of rational belief (or, as we shall later call them, probabilities) about the truth value of propositions are represented by real numbers.
  - (Transitivity). If the belief in A is greater than the belief in B, and the belief in B is greater than the belief in C, then the belief in A must be greater than the belief in C.
  - (Consistency). If the belief in an event can be inferred in two different ways, then the two ways must agree on the resulting belief.

- This effort resulted in confirming that the **sum and product rules of Probability Theory** (to be discussed below) are the **only** proper rational way to process belief intensities.

- $\Rightarrow$ Probability theory (PT) provides the **theory of optimal processing of incomplete information** (see Cox theorem, and Caticha, pp.7-24).

Why Probability Theory for Machine Learning?

- Machine learning concerns updating our beliefs about appropriate settings for model parameters from new information (namely a data set), and therefore PT provides the *optimal calculus for machine learning*.

- In general, nearly all interesting questions in machine learning (and information processing in general) can be stated in the following form (a conditional probability):

$$p(\texttt{whatever-we-want-to-know} \,|\, \texttt{whatever-we-do-know})$$

where $p(a|b)$ means the probability that a is true, given that b is true.

- Examples
  - Predictions

$$p(\,\texttt{future-observations} \,|\, \texttt{past-observations}\,)$$

  - Classify a received data point x

$$p(\,x\texttt{-belongs-to-class-}k \,|\, x\,)$$

  - Update a model based on a new observation

$$p(\,\texttt{model-parameters} \,|\, \texttt{new-observation}, \texttt{past-observations}\,)$$

Frequentist vs. Bayesian Interpretation of Probabilities

- The interpretation of a probability as a **degree-of-belief** about the truth value of an event is also called the **Bayesian** interpretation.

- In the **Bayesian** interpretation, the probability is associated with a **state-of-knowledge** (usually held by a person, but formally by a rational agent).
  - For instance, in a coin tossing experiment, $p(\texttt{tail}) = 0.4$ should be interpreted as the belief that there is a 40% chance that $\texttt{tail}$ comes up if the coin were tossed.
  - Under the Bayesian interpretation, PT calculus (sum and product rules) **extends boolean logic to rational reasoning with uncertainty**.

- The Bayesian interpretation contrasts with the **frequentist** interpretation of a probability as the relative frequency that an event would occur under repeated execution of an experiment.
  - For instance, if the experiment is tossing a coin, then $p(\texttt{tail}) = 0.4$ means that in the limit of a large number of coin tosses, 40% of outcomes turn up as $\texttt{tail}$.

- The Bayesian viewpoint is more generally applicable than the frequentist viewpoint, e.g., it is hard to apply the frequentist viewpoint to events like '$\texttt{it will rain tomorrow}$'.

- The Bayesian viewpoint is clearly favored in the machine learning community. (In this class, we also strongly favor the Bayesian interpretation).

- Aubrey Clayton, in his wonderful book Bernoulli's fallacy (2021), writes about this issue:

> "Compared with Bayesian methods, standard [frequentist] statistical techniques use only a small fraction of the available information about a research hypothesis (how well it predicts some observation), so naturally they will struggle when that limited information proves inadequate. Using standard statistical methods is like driving a car at night on a poorly lit highway:

to keep from going in a ditch, we could build an elaborate system of bumpers and guardrails and equip the car with lane departure warnings and sophisticated navigation systems, and even then we could at best only drive to a few destinations. Or we could turn on the headlights."

- In this class, we aim to turn on the headlights and illuminate the elegance and power of the Bayesian approach to information processing.

Events

- Technically, a probability expresses a degree-of-belief in the truth value of an event. Let's first define an event.
- We define an **event** A as a statement, whose truth can be contemplated by a person, e.g.,

$$A = \text{`it will rain tomorrow'}$$

- We write the denial of A, i.e. the event **not**-A, as $\bar{A}$.
- Events can be logically combined to create new events. Given two events A and B, we will shortly write the **conjunction** (logical-and) $A \wedge B$ as A, B or AB. The conjunction AB is true only if both A and B are true.
- We will write the **disjunction** (logical-or) $A \vee B$ also as $A + B$, which is true if either A or B is true or both A and B are true. (Note that the plus-sign is not an arithmetic here but rather a logical operator to process truth values).

Probability

- For any event A, with background knowledge I, the **conditional probability of A given I**, is written as

$$p(A|I)\,.$$

- $p(A|I)$ indicates the degree-of-belief in event A, given that I is true.
- In principle, all probabilities are conditional probabilities of the type $p(A|I)$, since there is always some background knowledge. However, we often write $p(A)$ rather than $p(A|I)$ if the background knowledge I is assumed to be obviously present. E.g., we usually write $p(A)$ rather than $p(\,A\,|\,\text{the-sun-comes-up-tomorrow}\,)$.
- The expression $p(A, B)$ is called the **joint probability** of events A and B. Note that $p(A, B) = p(B, A)$, since $AB = BA$. Therefore the order of arguments in a joint probability distribution does not matter: $p(A, B, C, D) = p(C, A, D, B)$, etc.
- Note that, if X is a variable, then an *assignment* $X = x$ (where x is a value, e.g., $X = 5$) can be interpreted as an event. Hence, the expression $p(X = 5)$ should be interpreted as the degree-of-belief that variable X takes on the value 5.
- If X is a *discretely* valued variable, then $p(X = x)$ is a probability *mass* function (PMF) with $0 \leq p(X = x) \leq 1$ and normalization $\sum_x p(x) = 1$.
- If X is *continuously* valued, then $p(X = x)$ is a probability *density* function (PDF) with $p(X = x) \geq 0$ and normalization $\int_x p(x)\mathrm{d}x = 1$.
  - Note that if X is continuously valued, then the value of $p(x)$ is not necessarily ≤ 1. E.g., a uniform distribution on the continuous domain $[0, .5]$ has value $p(x) = 2$.
- The notational conventions in PT are unfortunately a bit sloppy:(For instance, in the context of a variable X, we often write $p(x)$ rather than $p(X = x)$, assuming that the reader understands the context.

Probability Theory Calculus

- The following product and sum rules are also known as the **axioms of probability theory**, but as discussed above, under some mild assumptions, they can be derived as the unique rules for *rational reasoning under uncertainty* (Cox theorem, 1946, and Caticha, 2012, pp.7-26).
- **Sum rule**. The disjunction of two events A and B with given background I is

$$\boxed{p(A + B|I) = p(A|I) + p(B|I) - p(A, B|I)}$$

- **Product rule**. The conjuction of two events A and B with given background I is

$$\boxed{p(A,B|I) = p(A|B,I)\,p(B|I)}$$

- PT extends (propositional) logic in reasoning about the truth value of events. Logic reasons about the truth value of events on the binary domain {0,1} (FALSE and TRUE), whereas PT extends the range to degrees-of-belief, represented by real numbers in [0,1].

- **All legitimate probabilistic relations can be derived from the sum and product rules!**

Independent, Exclusive and Exhaustive Events

- It will be helpful to introduce some terms concerning special relationships between events.

- Two events A and B are said to be **independent** if the probability of one event is not altered by information about the truth of the other event, i.e.,

$$p(A|B) = p(A)\,.$$

  - $\Rightarrow$ If A and B are independent, then the product rule simplifies to

$$p(A,B) = p(A)p(B)\,.$$

  - A and B with given background I are said to be **conditionally independent** if $p(A|B,I) = p(A|I)$. In that case, the product rule simplifies to $p(A,B|I) = p(A|I)p(B|I)$.

- Two events A_1 and A_2 are said to be **mutually exclusive** ('disjoint') if they cannot be true simultanously, i.e., if $p(A_1,A_2) = 0$.
  - $\Rightarrow$ For mutually exclusive events, probability adds (this follows from the sum rule):

$$p(A_1 + A_2) = p(A_1) + p(A_2)$$

- A set of events $A_1, A_2, \ldots, A_N$ is said to be **collectively exhaustive** if one of the statements is necessarily true, i.e., $A_1 + A_2 + \cdots + A_N = \mathrm{TRUE}$, or equivalently

$$p(A_1 + A_2 + \cdots + A_N) = 1$$

- If a set of events $A_1, A_2, \ldots, A_n$ are both **mutually exclusive** and **collectively exhausitive** events, then we say that they **partition the universe**. Technically, this means that

$$\sum_{n=1}^{N} p(A_n) = p(A_1 + \ldots + A_N) = 1$$

- We mentioned before that every inference problem in PT can be evaluated through the sum and product rules. Next, we present two useful corollaries: (1) *Marginalization* and (2) *Bayes rule*.

Marginalization

- Let A and $B_1, B_2, \ldots, B_n$ be events, where $B_1, B_2, \ldots, B_n$ partitions the universe. Then

$$\sum_{i=1}^{n} p(A,B_i) = p(A)\,.$$

- This rule is called the law of total probability. Proof:

$$\begin{aligned}
\sum_i p(A,B_i) &= p(\sum_i AB_i) && (\text{since all } AB_i \text{ are disjoint})\\
&= p(A,\sum_i B_i)\\
&= p(A,\mathrm{TRUE}) && (\text{since } B_i \text{ are exhaustive})\\
&= p(A)
\end{aligned}$$

- A very practical application of this law is to get rid of a variable that we are not interested in. For instance, if X and $Y \in \{y_1, y_2, \ldots, y_n\}$ are discrete variables, then

$$p(X) = \sum_{i=1}^{n} p(X, Y = y_i)\,.$$

- Summing Y out of a joint distribution $p(X, Y)$ is called **marginalization** and the result $p(X)$ is sometimes referred to as the **marginal probability** of X.

- Note that marginalization can be understood as applying a "generalized" sum rule. Bishop (p.14) and some other authors also refer to this as the sum rule, but we do not follow that terminology.

- Of course, in the continuous domain, marginalization becomes

$$p(X) = \int_Y p(X, Y)\,\mathrm{d}Y$$

Bayes Rule

- Consider two variables D and θ. It follows from symmetry arguments that $p(D, \theta) = p(\theta, D)$, and hence that

$$p(D|\theta)p(\theta) = p(\theta|D)p(D)$$

  or, equivalently,

$$p(\theta|D) = \frac{p(D|\theta)}{p(D)}p(\theta)\,. \qquad \text{(Bayes rule)}$$

- This last formula is called **Bayes rule**, named after its inventor Thomas Bayes (1701-1761). While Bayes rule is always true, a particularly useful application occurs when D refers to an observed data set and θ is set of unobserved model parameters. In that case,
  - the **prior** probability $p(\theta)$ represents our **state-of-knowledge** about proper values for θ, before seeing the data D.
  - the **posterior** probability $p(\theta|D)$ represents our state-of-knowledge about θ after we have seen the data.

$\Rightarrow$ Bayes rule tells us how to update our knowledge about model parameters when facing new data. Hence,

Bayes rule is the fundamental rule for learning from data!

Bayes Rule Nomenclature

- Some nomenclature associated with Bayes rule:

$$\underbrace{p(\theta|D)}_{\text{posterior}} = \frac{\overbrace{p(D|\theta)}^{\text{likelihood}} \times \overbrace{p(\theta)}^{\text{prior}}}{\underbrace{p(D)}_{\text{evidence}}}$$

- Note that the evidence (a.k.a. *marginal likelihood*) can be computed from the numerator through marginalization since

$$p(D) = \int p(D, \theta)\,\mathrm{d}\theta = \int p(D|\theta)\,p(\theta)\,\mathrm{d}\theta$$

- Hence, having access to likelihood and prior is in principle sufficient to compute both the evidence and the posterior. To emphasize that point, Bayes rule is sometimes written as a transformation:

$$\underbrace{\underbrace{p(\theta|D)}_{\text{posterior}} \cdot \underbrace{p(D)}_{\text{evidence}}}_{\text{this is what we want to compute}} = \underbrace{\underbrace{p(D|\theta)}_{\text{likelihood}} \cdot \underbrace{p(\theta)}_{\text{prior}}}_{\text{this is available}}$$

- For a given data set D, the posterior probabilities of the parameters scale relatively against each other as

$$p(\theta|D) \propto p(D|\theta)p(\theta)$$

- $\Rightarrow$ All that we can learn from the observed data is contained in the likelihood function $p(D|\theta)$. This is called the **likelihood principle**.

The Likelihood Function vs the Sampling Distribution

- Consider a distribution $p(D|\theta)$, where D relates to variables that are observed (i.e., a "data set") and θ are model parameters.

- In general, $p(D|\theta)$ is just a function of the two variables D and θ. We distinguish two interpretations of this function, depending on which variable is observed (or given by other means).

- The **sampling distribution** (a.k.a. the **data-generating** distribution)

$$p(D|\theta = \theta_0)$$

  (which is a function of D only) describes a probability distribution for data D, assuming that it is generated by the given model with parameters fixed at $\theta = \theta_0$.

- In a machine learning context, often the data is observed, and θ is the free variable. In that case, for given observations $D = D_0$, the **likelihood function** (which is a function only of the model parameters θ) is defined as

$$\mathrm{L}(\theta) \triangleq p(D = D_0|\theta)$$

- Note that $\mathrm{L}(\theta)$ is not a probability distribution for θ since in general $\sum_\theta \mathrm{L}(\theta) \neq 1$.

Code Example: Sampling Distribution and Likelihood Function for the Coin Toss

- Consider the following simple model for the outcome (head or tail) $y \in \{0, 1\}$ of a biased coin toss with parameter $\theta \in [0, 1]$:

$$p(y|\theta) = \theta^y(1-\theta)^{1-y}$$

- Next, we use Julia to plot both the sampling distribution

$$p(y|\theta = 0.5) = \begin{cases} 0.5 & \text{if } y = 0 \\ 0.5 & \text{if } y = 1 \end{cases}$$

  and the likelihood function

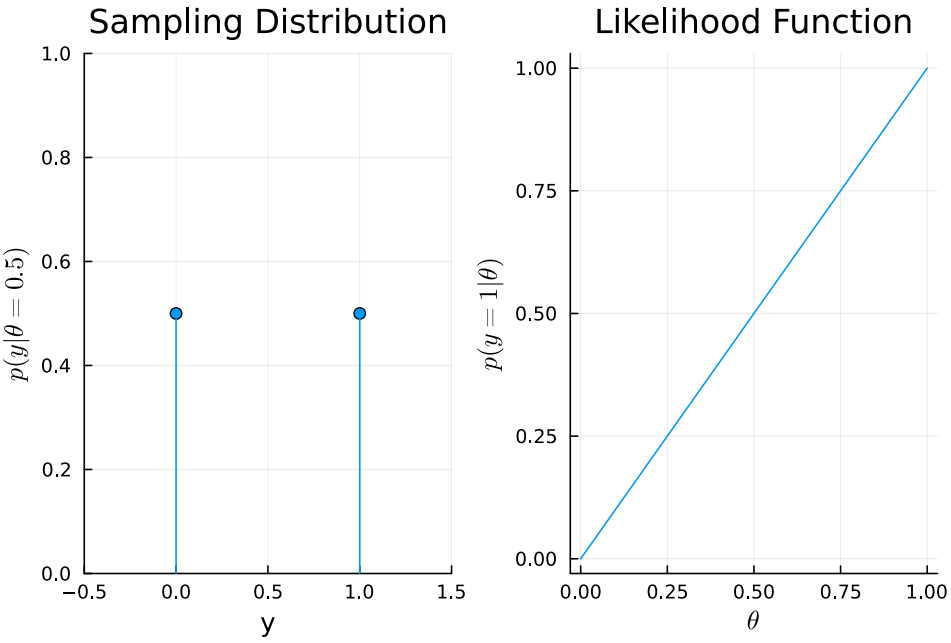
$$L(\theta) = p(y = 1|\theta) = \theta\,.$$

In [1]:
```
using Pkg; Pkg.activate("../."); Pkg.instantiate();
using IJulia; try IJulia.clear_output(); catch _ end
```

Out[1]:0

In [2]:
```
using Plots
using LaTeXStrings

f(y,θ) = θ.^y .* (1 .- θ).^(1 .- y) # p(y|θ)

θ = 0.5
p1 = plot([0,1], f([0,1], θ),
  line=:stem, marker=:circle, xrange=(-0.5, 1.5), yrange=(0,1), title="Sampling Distribution", xlabel="y", y

_θ = 0:0.01:1
y=1
p2 = plot(_θ, f(y, _θ),
  ylabel=L"p(y=%$y | θ)", xlabel=L"θ", title="Likelihood Function", label="")

plot(p1, p2)
```

Out[2]:



- The (discrete) sampling distribution is a valid probability distribution. However, the likelihood function $L(\theta)$ clearly isn't, since $\int_0^1 L(\theta)\mathrm{d}\theta \neq 1$.

Probabilistic Inference

- **Probabilistic inference** refers to computing

$$p(\text{ whatever-we-want-to-know } | \text{ whatever-we-already-know })$$

  - For example:

$$p(\text{ Mr.S.-killed-Mrs.S. } | \text{ he-has-her-blood-on-his-shirt })$$
$$p(\text{ transmitted-codeword } | \text{ received-codeword })$$

- This can be accomplished by repeated application of sum and product rules.
- In particular, consider a joint distribution $p(X, Y, Z)$. Assume we are interested in $p(X|Z)$:

$$p(X|Z) \overset{p}{=} \frac{p(X,Z)}{p(Z)} \overset{s}{=} \frac{\sum_Y p(X,Y,Z)}{\sum_{X,Y} p(X,Y,Z)},$$

  where the 's' and 'p' above the equality sign indicate whether the sum or product rule was used.

- In the rest of this course, we'll encounter many long probabilistic derivations. For each manipulation, you should be able to associate an 's' (for sum rule), a 'p' (for product or Bayes rule) or an 'm' (for a simplifying model assumption) above any equality sign.

Revisiting the Challenge: Disease Diagnosis

- **Problem**: Given a disease D with prevalence of 1% and a test procedure T with sensitivity ('true positive' rate) of 95% and specificity ('true negative' rate) of 85%, what is the chance that somebody who tests positive actually has the disease?
- **Solution**: The given information is $p(D = 1) = 0.01$, $p(T = 1|D = 1) = 0.95$ and $p(T = 0|D = 0) = 0.85$. We are asked to derive $p(D = 1|T = 1)$. We just follow the sum and product rules to derive the requested probability:

$$
\begin{aligned}
&p(D=1|T=1) \\
&\quad \underline{\overset{p}{=}} \frac{p(T=1,D=1)}{p(T=1)} \\
&\quad \underline{\overset{p}{=}} \frac{p(T=1|D=1)p(D=1)}{p(T=1)} \\
&\quad \underline{\overset{s}{=}} \frac{p(T=1|D=1)p(D=1)}{p(T=1|D=1)p(D=1)} \\
&\qquad + p(T=1|D=0)p(D=0) \\
&\quad = \frac{0.95 \times 0.01}{0.95 \times 0.01 + 0.15 \times 0.99} = 0.0601
\end{aligned}
$$

- Note that $p(\text{sick}|\text{positive test}) = 0.06$ while $p(\text{positive test}|\text{sick}) = 0.95$. This is a huge difference that is sometimes called the "medical test paradox" or the base rate fallacy.
- Many people have trouble distinguishing $p(A|B)$ from $p(B|A)$ in their heads. This has led to major negative consequences. For instance, unfounded convictions in the legal arena and even lots of unfounded conclusions in the pursuit of scientific results. See Ioannidis (2005) and Clayton (2021).

Inference Exercise: Bag Counter

- **Problem**: A bag contains one ball, known to be either white or black. A white ball is put in, the bag is shaken, and a ball is drawn out, which proves to be white. What is now the chance of drawing a white ball?
- **Solution**: Again, use Bayes and marginalization to arrive at $p(\text{white}|\text{data}) = 2/3$, see the Exercises notebook.
- $\Rightarrow$ Note that probabilities describe **a person's state of knowledge** rather than a 'property of nature'.

Inference Exercise: Causality?

- **Problem**: A dark bag contains five red balls and seven green ones. (a) What is the probability of drawing a red ball on the first draw? Balls are not returned to the bag after each draw. (b) If you know that on the second draw the ball was a green one, what is now the probability of drawing a red ball on the first draw?
- **Solution**: (a) $5/12$. (b) $5/11$, see the Exercises notebook.
- $\Rightarrow$ Again, we conclude that conditional probabilities reflect **implications for a state of knowledge** rather than temporal causality.

Moments of the PDF

- Distributions can often usefully be summarized by a set of values known as moments of the distribution.
- Consider a distribution $p(x)$. The first moment, also known as **expected value** or **mean** of $p(x)$ is defined as

$$
\mu_x = \mathbb{E}[x] \triangleq \int x\, p(x)\, \mathrm{d}x
$$

- The second central moment, also known as **variance** of x is defined as

$$
\Sigma_x \triangleq \mathbb{E}\left[(x-\mu_x)(x-\mu_x)^T\right]
$$

- The **covariance** matrix between *vectors* x and y is a mixed central moment, defined as

$$
\begin{aligned}
\Sigma_{xy} &\triangleq \mathbb{E}\left[(x-\mu_x)(y-\mu_y)^T\right] \\
&= \mathbb{E}\left[(x-\mu_x)(y^T-\mu_y^T)\right] \\
&= \mathbb{E}[xy^T] - \mu_x\mu_y^T
\end{aligned}
$$

- Clearly, if x and y are independent, then $\Sigma_{xy} = 0$, since in that case $\mathbb{E}[xy^T] = \mathbb{E}[x]\mathbb{E}[y^T] = \mu_x\mu_y^T$.

- Exercise: Proof that $\Sigma_{xy} = \Sigma_{yx}^T$ (making use of $(AB)^T = B^T A^T$).

Linear Transformations

- Consider an arbitrary distribution $p(X)$ with mean μ_x and variance Σ_x and the linear transformation

$$Z = AX + b\,.$$

- No matter the specification of $p(X)$, we can derive that (see Exercises notebook)

$$\begin{aligned} \mu_z &= A\mu_x + b && \text{(SRG-3a)} \\ \Sigma_z &= A\,\Sigma_x\,A^T && \text{(SRG-3b)} \end{aligned}$$

  - (The tag (SRG-3a) refers to the corresponding eqn number in Sam Roweis' Gaussian identities notes.)

PDF for the Sum of Two Variables

- Given eqs SRG-3a and SRG-3b (previous cell), you should now be able to derive the following: for any distribution of variable X and Y and sum $Z = X + Y$ (proof by Exercise)

$$\begin{aligned} \mu_z &= \mu_x + \mu_y \\ \Sigma_z &= \Sigma_x + \Sigma_y + 2\Sigma_{xy} \end{aligned}$$

- Clearly, it follows that if X and Y are **independent**, then

$$\Sigma_z = \Sigma_x + \Sigma_y$$

- More generally, assume two jointly continuous variables X and Y, with joint PDF $p_{xy}(x, y)$. Let $Z = X + Y$, then

$$\begin{aligned} \mathrm{Prob}(Z \le z) &= \mathrm{Prob}(X + Y \le z) \\ &= \int_{-\infty}^{\infty} \left(\int_{-\infty}^{z-x} p_{xy}(x, y) \mathrm{d}y \right) \mathrm{d}x \\ &= \int_{-\infty}^{\infty} \left(\int_{-\infty}^{z} p_{xy}(x, t-x) \mathrm{d}t \right) \mathrm{d}x \\ &= \int_{-\infty}^{z} \Bigg(\underbrace{\int_{-\infty}^{\infty} p_{xy}(x, t-x) \mathrm{d}x}_{p_z(t)} \Bigg) \mathrm{d}t \end{aligned}$$

- Hence, the PDF for the sum Z is given by $p_z(z) = \int_{-\infty}^{\infty} p_{xy}(x, z-x) \mathrm{d}x$.

- In particular, if X and Y are **independent** variables, then

$$\begin{aligned} p_z(z) = \int_{-\infty}^{\infty} p_x(x) p_y(z-x)\,\mathrm{d}x &= p_x(z) \\ &* p_y(z)\,, \end{aligned}$$

  which is the **convolution** of the two marginal PDFs.

- https://en.wikipedia.org/wiki/List_of_convolutions_of_probability_distributions shows how these convolutions work out for a few common probability distributions.

Code Example: Sum of Two Gaussian Distributed Variables

- Consider two independent Gaussian-distributed variables X and Y (see wiki:normal-distribution for definition of a Gaussian (=Normal) distribution):

$$\begin{aligned} p_X(x) &= \mathcal{N}(\,x \,|\, \mu_X, \sigma_X^2\,) \\ p_Y(y) &= \mathcal{N}(\,y \,|\, \mu_Y, \sigma_Y^2\,) \end{aligned}$$

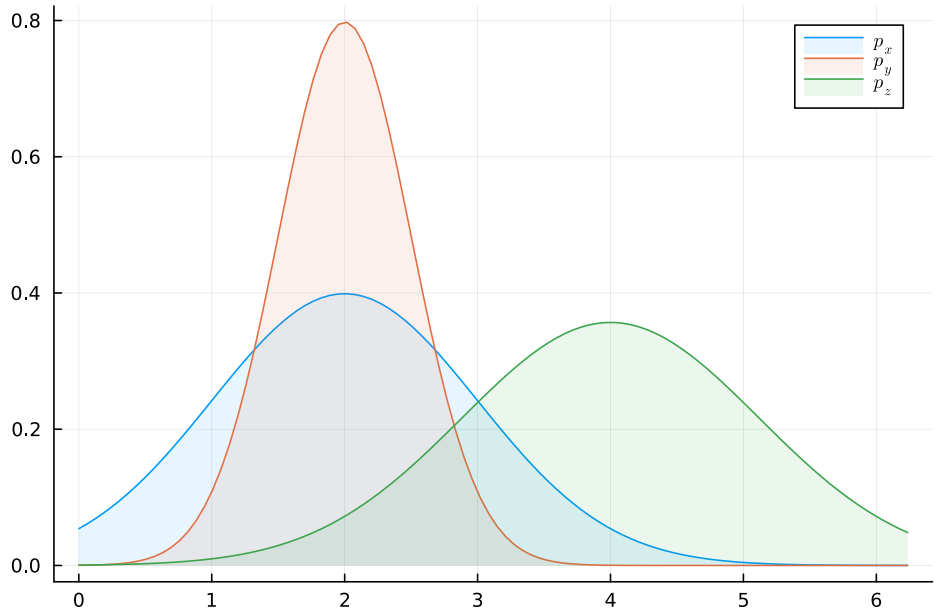
- Let $Z = X + Y$. Performing the convolution (nice exercise) yields a Gaussian PDF for Z:

$$p_Z(z) = \mathcal{N}(\, z \,|\, \mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2 \,).$$

- We illustrate the distributions for X, Y and Z using Julia:

In [3]:
```
using Plots, Distributions, LaTeXStrings

μx = 2.
σx = 1.
μy = 2.
σy = 0.5
μz = μx+μy; σz = sqrt(σx^2 + σy^2)
x = Normal(μx, σx)
y = Normal(μy, σy)
z = Normal(μz, σz)
range_min = minimum([μx-2*σx, μy-2*σy, μz-2*σz])
range_max = maximum([μx+2*σx, μy+2*σy, μz+2*σz])
range_grid = range(range_min, stop=range_max, length=100)
plot(range_grid, pdf.(x,range_grid), label=L"p_x", fill=(0, 0.1))
plot!(range_grid, pdf.(y,range_grid), label=L"p_y", fill=(0, 0.1))
plot!(range_grid, pdf.(z,range_grid), label=L"p_z", fill=(0, 0.1))
```

Out[3]:



PDF for the Product of Two Variables

- For two continuous **independent** variables X and Y, with PDF's $p_x(x)$ and $p_y(y)$, the PDF of $Z = XY$ is given by

$$p_z(z) = \int_{-\infty}^{\infty} p_x(x)\, p_y(z/x)\, \frac{1}{|x|}\, \mathrm{d}x\,.$$

- For proof, see https://en.wikipedia.org/wiki/Product_distribution.

- Generally, this integral does not lead to an analytical expression for $p_z(z)$.

- For example, the product of two independent variables that are both Gaussian-distributed does not lead to a Gaussian distribution.
  - Exception: the distribution of the product of two variables that both have log-normal distributions is again a lognormal distribution. (If X has a normal distribution, then $Y = \exp(X)$ has a log-normal distribution.)

Variable Transformations

- Suppose x is a **discrete** variable with probability **mass** function $P_x(x)$, and $y = h(x)$ is a one-to-one function with $x = g(y) = h^{-1}(y)$. Then

$$P_y(y) = P_x(g(y))\,.$$

- **Proof**:

$$P_y(\hat{y}) = P(y = \hat{y}) = P(h(x) = \hat{y})$$
$$= P(x = g(\hat{y})) = P_x(g(\hat{y})). \square$$

- If x is defined on a **continuous** domain, and $p_x(x)$ is a probability **density** function, then probability mass is represented by the area under a (density) curve. Let $a = g(c)$ and $b = g(d)$. Then

$$\begin{aligned} P(a \leq x \leq b) &= \int_a^b p_x(x)\mathrm{d}x \\ &= \int_{g(c)}^{g(d)} p_x(x)\mathrm{d}x \\ &= \int_c^d p_x(g(y))\mathrm{d}g(y) \\ &= \int_c^d \underbrace{p_x(g(y))g'(y)}_{p_y(y)}\mathrm{d}y \\ &= P(c \leq y \leq d) \end{aligned}$$

- Equating the two probability masses leads to identification of the relation

$$p_y(y) = p_x(g(y))g'(y)\,,$$

  which is also known as the Change-of-Variable theorem.

- If the tranformation $y = h(x)$ is not invertible, then $x = g(y)$ does not exist. In that case, you can still work out the transformation by equating equivalent probability masses in the two domains.

Example: Transformation of a Gaussian Variable

- Let $p_x(x) = \mathcal{N}(x|\mu, \sigma^2)$ and $y = \frac{x-\mu}{\sigma}$.
- **Problem**: What is $p_y(y)$?
- **Solution**: Note that $h(x)$ is invertible with $x = g(y) = \sigma y + \mu$. The change-of-variable formula leads to

$$\begin{aligned} p_y(y) &= p_x(g(y)) \cdot g'(y) \\ &= p_x(\sigma y + \mu) \cdot \sigma \\ &= \frac{1}{\sigma\sqrt{(2\pi)}}\exp\left(-\frac{(\sigma y + \mu - \mu)^2}{2\sigma^2}\right) \cdot \sigma \\ &= \frac{1}{\sqrt{(2\pi)}}\exp\left(-\frac{y^2}{2}\right) \\ &= \mathcal{N}(y|0, 1) \end{aligned}$$

A Notational Convention

- Finally, here is a notational convention that you should be precise about (but many authors are not).
- If you want to write that a variable x is distributed as a Gaussian with mean μ and covariance matrix Σ, you can write this properly in either of two ways:

$$\begin{aligned} p(x) &= \mathcal{N}(x|\mu, \Sigma) \\ x &\sim \mathcal{N}(\mu, \Sigma) \end{aligned}$$

- In the second version, the symbol $\sim$ can be interpreted as "is distributed as" (a Gaussian with parameters μ and Σ).
- Don't write $p(x) = \mathcal{N}(\mu, \Sigma)$ because $p(x)$ is a function of x but $\mathcal{N}(\mu, \Sigma)$ is not.
- Also, $x \sim \mathcal{N}(x|\mu, \Sigma)$ is not proper because you already named the argument at the right-hand-site. On the other hand, $x \sim \mathcal{N}(\cdot|\mu, \Sigma)$ is fine, as is the shorter $x \sim \mathcal{N}(\mu, \Sigma)$.

Summary

- Probabilities should be interpretated as degrees of belief, i.e., a state-of-knowledge, rather than a property of nature.
- We can do everything with only the **sum rule** and the **product rule**. In practice, **Bayes rule** and **marginalization** are often very useful for inference, i.e., for computing

$$p(\text{ what-we-want-to-know} \mid \text{what-we-already-know }).$$

- Bayes rule

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)}$$

  is the fundamental rule for learning from data!
- For a variable X with distribution $p(X)$ with mean μ_x and variance Σ_x, the mean and variance of the **Linear Transformation** $Z = AX + b$ is given by

$$\mu_z = A\mu_x + b \tag{SRG-3a}$$
$$\Sigma_z = A\,\Sigma_x\,A^T \tag{SRG-3b}$$

- That's really about all you need to know about probability theory, but you need to *really* know it, so do the Exercises!

Bayesian Machine Learning

Preliminaries

- Goals
  - Introduction to Bayesian (i.e., probabilistic) modeling
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 68-74 (on the coin toss example)
    - Ariel Caticha - 2012 - Entropic Inference and the Foundations of Physics, pp.35-44 (section 2.9, on deriving Bayes rule for updating probabilities)

Challenge: Predicting a Coin Toss

- **Problem**: We observe the following sequence of heads (outcome $= 1$) and tails (outcome $= 0$) when tossing the same coin repeatedly

$$D = \{1011001\}.$$

- What is the probability that heads comes up next?
- **Solution**: later in this lecture.

The Bayesian Machine Learning Framework

- Suppose that your application is to predict a future observation x, based on N past observations $D = \{x_1, \ldots, x_N\}$.
- The Bayesian design approach to solving this task involves four stages:

```
 REPEAT     1- Model specification     2- Parameter estimation     3- Model evaluation   UNTIL model
performance is satisfactory     4- Apply model
```

- In principle, based on the model evaluation results, you may want to re-specify your model and *repeat* the design process (a few times), until model performance is acceptable.

- Next, we discuss these four stages in a bit more detail.

(1) Model specification

- Your first task is to propose a probabilistic model for generating the observations x.
- A probabilistic model m consists of a joint distribution $p(x, \theta|m)$ that relates observations x to model parameters θ. Usually, the model is proposed in the form of a data generating distribution $p(x|\theta, m)$ and a prior $p(\theta|m)$.
- *You* are responsible to choose the data generating distribution $p(x|\theta)$ based on your physical understanding of the data generating process. (For brevity, if we are working on one given model m with no alternative models, we usually drop the given dependency on m from the notation).
- *You* must also choose the prior $p(\theta)$ to reflect what you know about the parameter values before you see the data D.

(2) Parameter estimation

- Note that, for a given data set $D = \{x_1, x_2, \ldots, x_N\}$ with *independent* observations x_n, the likelihood factorizes as

$$p(D|\theta) = \prod_{n=1}^{N} p(x_n|\theta)\,,$$

  so usually you select a model for generating one observation x_n and then use (in-)dependence assumptions to combine these models into a likelihood function for the model parameters.

- The likelihood and prior both contain information about the model parameters. Next, you use Bayes rule to fuse these two information sources into a posterior distribution for the parameters:

$$\begin{aligned}\underbrace{p(\theta|D)}_{\text{posterior}} &= \frac{p(D|\theta)p(\theta)}{p(D)} \\ &= \frac{p(D|\theta)p(\theta)}{\int p(D|\theta)p(\theta)\mathrm{d}\theta}\end{aligned}$$

- Note that there's **no need for you to design some clever parameter estimation algorithm**. Bayes rule *is* the parameter estimation algorithm, which can be entirely expressed in terms of the likelihood and prior. The only complexity lies in the computational issues!
- This parameter estimation "recipe" works if the right-hand side (RHS) factors can be evaluated; the computational details can be quite challenging and this is what machine learning is about.
- $\Rightarrow$ **Machine learning is EASY, apart from computational details :)**

(3) Model Evaluation

- In the framework above, parameter estimation was executed by "perfect" Bayesian reasoning. So is everything settled now?
- No, there appears to be one remaining problem: how good really were our model assumptions $p(x|\theta)$ and $p(\theta)$? We want to "score" the model performance.
- Note that this question is only interesting in practice if we have alternative models to choose from. After all, if you don't have an alternative model, any value for the model evidence would still not lead you to switch to another model.
- Let's assume that we have more candidate models, say $\mathcal{M} = \{m_1, \ldots, m_K\}$ where each model relates to specific prior $p(\theta|m_k)$ and likelihood $p(D|\theta, m_k)$? Can we evaluate the relative performance of a model against another model from the set?
- Start again with **model specification**. *You* must now specify a prior $p(m_k)$ (next to the likelihood $p(D|\theta, m_k)$ and prior $p(\theta|m_k)$) for each of the models and then solve the desired inference problem:

$$\begin{aligned}
\underbrace{p(m_k|D)}_{\substack{\text{model}\\\text{posterior}}} &= \frac{p(D|m_k)p(m_k)}{p(D)} \\
&\propto p(m_k)\cdot p(D|m_k) \\
&= p(m_k)\cdot\int_\theta p(D,\theta|m_k)\,\mathrm{d}\theta \\
&= \underbrace{p(m_k)}_{\substack{\text{model}\\\text{prior}}}\cdot\underbrace{\int_\theta \underbrace{p(D|\theta,m_k)}_{\text{likelihood}}\,\underbrace{p(\theta|m_k)}_{\text{prior}}\,\mathrm{d}\theta}_{\substack{\text{evidence } p(D|m_k)\\=\text{ model likelihood}}}
\end{aligned}$$

- You *can* evaluate the RHS of this equation since you selected the model priors $p(m_k)$, the parameter priors $p(\theta|m_k)$ and the likelihoods $p(D|\theta,m_k)$.
- You can now compare posterior distributions $p(m_k|D)$ for a set of models $\{m_k\}$ and decide on the merits of each model relative to alternative models. This procedure is called **Bayesian model comparison**.
- Note that, to evaluate the model posterior, you must calculate the "model evidence" $p(D|m_k)$, which can be interpreted as a likelihood function for model m_k.
- $\Rightarrow$ In a Bayesian framework, **model estimation** follows the same recipe as parameter estimation; it just works at one higher hierarchical level. Compare the required calulations:

$$\begin{aligned}
p(\theta|D) &\propto p(D|\theta)p(\theta) && \text{(parameter estimation)}\\
p(m_k|D) &\propto p(D|m_k)p(m_k) && \text{(model comparison)}
\end{aligned}$$

- Again, **no need to invent a special algorithm for estimating the performance of your model**. Straightforward application of probability theory takes care of all that.
- In principle, you could proceed with asking how good your choice for the candidate model set $\mathcal{M}$ was. You would have to provide a set of alternative model sets $\{\mathcal{M}_1,\mathcal{M}_2,\ldots,\mathcal{M}_M\}$ with priors $p(\mathcal{M}_m)$ for each set and compute posteriors $p(\mathcal{M}_m|D)$. And so forth ...
- With the (relative) performance evaluation scores of your model in hand, you could now re-specify your model (hopefully an improved model) and *repeat* the design process until the model performance score is acceptable.

Bayes Factors

- As an aside, in the (statistics and machine learning) literature, performance comparison between two models is often reported by the Bayes Factor, which is defined as the ratio of model evidences:

$$\begin{aligned}
\underbrace{\frac{p(D|m_1)}{p(D|m_2)}}_{\text{Bayes Factor}} &= \frac{\frac{p(D,m_1)}{p(m_1)}}{\frac{p(D,m_2)}{p(m_2)}} \\
&= \frac{p(D,m_1)}{p(m_1)}\cdot\frac{p(m_2)}{p(D,m_2)} \\
&= \frac{p(m_1|D)p(D)}{p(m_1)}\cdot\frac{p(m_2)}{p(m_2|D)p(D)} \\
&= \underbrace{\frac{p(m_1|D)}{p(m_2|D)}}_{\substack{\text{posterior}\\\text{ratio}}}\cdot\underbrace{\frac{p(m_2)}{p(m_1)}}_{\substack{\text{prior}\\\text{ratio}}}
\end{aligned}$$

  - Hence, for equal model priors ($p(m_1)=p(m_2)=0.5$), the Bayes Factor reports the posterior probability ratio for the two models.
- In principle, any hard decision on which is the better model has to accept some *ad hoc* arguments, but Jeffreys (1961) advises the following interpretation of the log-Bayes factor

$$\log_{10} B_{12} = \log_{10} \frac{p(D|m_1)}{p(D|m_2)}$$

$\log_{10} B_{12}$	**Evidence for** m_1
0 to 0.5	not worth mentioning
0.5 to 1	substantial
1 to 2	strong
>2	decisive

(4) Prediction

- Once we are satisfied with the evidence for a (trained) model, we can apply the model to our prediction/classification/etc task.
- Given the data D, our knowledge about the yet unobserved datum x is captured by (everything is conditioned on the selected model)

$$\begin{aligned} p(x|D) &\overset{s}{=} \int p(x, \theta|D)\, \mathrm{d}\theta \\ &\overset{p}{=} \int p(x|\theta, D) p(\theta|D)\, \mathrm{d}\theta \\ &\overset{m}{=} \int \underbrace{p(x|\theta)}_{\text{data generation dist.}} \cdot \underbrace{p(\theta|D)}_{\text{posterior}}\, \mathrm{d}\theta \end{aligned}$$

- In the last equation, the simplification $p(x|\theta, D) = p(x|\theta)$ follows from our model specification. We assumed a *parametric* data generating distribution $p(x|\theta)$ with no explicit dependency on the data set D. The information from the data set D has been absorded in the posterior $p(\theta|D)$, so all information from D is passed to x through the (posterior distribution over the) parameters θ. Technically, x is conditionally independent from D, given the parameters θ.
- Again, **no need to invent a special prediction algorithm**. Probability theory takes care of all that. The complexity of prediction is just computational: how to carry out the marginalization over θ.
- Note that the application of the learned posterior $p(\theta|D)$ not necessarily has to be a prediction task. We use it here as an example, but other applications (e.g., classification, regression etc.) are of course also possible.

Prediction with multiple models

- When you have a posterior $p(m_k|D)$ for the models, you don't *need* to choose one model for the prediction task. You can do prediction by **Bayesian model averaging**, which combines the predictive power from all models:

$$\begin{aligned} p(x|D) &= \sum_k \int p(x, \theta, m_k|D)\, \mathrm{d}\theta \\ &= \sum_k \int p(x|\theta, m_k)\, p(\theta|m_k, D)\, p(m_k|D)\, \mathrm{d}\theta \\ &= \sum_k \underbrace{p(m_k|D)}_{\substack{\text{model} \\ \text{posterior}}} \cdot \int \underbrace{p(\theta|m_k, D)}_{\substack{\text{parameter} \\ \text{posterior}}}\, \underbrace{p(x|\theta, m_k)}_{\substack{\text{data generating} \\ \text{distribution}}}\, \mathrm{d}\theta \end{aligned}$$

- Alternatively, if you do need to work with one model (e.g. due to computational resource constraints), you can for instance select the model with largest posterior $p(m_k|D)$ and use that model for prediction. This is called **Bayesian model selection**.
- Bayesian model averaging is the principal way to apply PT to machine learning. You don't throw away information by discarding lesser performant models, but rather use PT (marginalization of models) to compute

$$p(\text{what-I-am-interested-in} \mid \\ \text{all available information})\,.$$

We're Done!

- In principle, you now have the recipe in your hands now to solve all your prediction/classification/regression etc problems by the same method:

  1. specify a model
  2. train the model (by PT)
  3. evaluate the model (by PT); if not satisfied, goto 1
  4. apply the model (by PT)

- Crucially, there is no need to invent clever machine learning algorithms, and there is no need to invent a clever prediction algorithm nor a need to invent a model performance criterion. Instead, you propose a model and, from there on, you let PT reason about everything that you care about.

- Your problems are only of computational nature. Perhaps the integral to compute the evidence may not be analytically tractable, etc.

Bayesian Evidence as a Model Performance Criterion

- I'd like to convince you that Bayesian model evidence is an excellent criterion for assessing your model's performance. To do so, let us consider a decomposition that relates model evidence to other highly-valued criteria such as **accuracy** and **model complexity**.

- Consider a model $p(x, \theta|m)$ and a data set $D = \{x_1, x_2, \ldots, x_N\}$.

- Given the data set D, the log-evidence for model m decomposes as follows (please check the derivation):

$$\begin{aligned}
\underbrace{\log p(D|m)}_{\text{log-evidence}} &= \log p(D|m) \cdot \underbrace{\int p(\theta|D,m)\mathrm{d}\theta}_{\text{evaluates to 1}} \\
&= \int p(\theta|D,m) \log p(D|m)\mathrm{d}\theta \\
&\quad (\text{move } \log p(D|m) \text{ into the integral}) \\
&= \int p(\theta|D,m) \log \underbrace{\frac{p(D|\theta,m)p(\theta|m)}{p(\theta|D,m)}}_{\text{by Bayes rule}}\mathrm{d}\theta \\
&= \underbrace{\int p(\theta|D,m) \log p(D|\theta,m)\mathrm{d}\theta}_{\text{accuracy (a.k.a. data fit)}} \\
&\quad - \underbrace{\int p(\theta|D,m) \log \frac{p(\theta|D,m)}{p(\theta|m)}\mathrm{d}\theta}_{\text{complexity}}
\end{aligned}$$

- The "accuracy" term (also known as data fit) measures how well the model predicts the data set D. We want this term to be high because good models should predict the data D well. Indeed, higher accuracy leads to higher model evidence. To achieve high accuracy, applying Bayes' rule will shift the posterior $p(\theta|D)$ away from the prior towards the likelihood function $p(D|\theta)$.

- The second term (complexity) is technically a Kullback-Leibler divergence (KLD) between the posterior and prior distributions, see OPTIONAL SLIDE. The KLD is an information-theoretic quantity that can be interpreted as a "distance" measure between two distributions. In other words, the complexity term measures how much the beliefs about θ changed, due to learning from the data D. Generally, we like the complexity term to be low, because moving away means forgetting previously acquired information represented by the prior. Indeed, lower complexity leads to higher model evidence.

- Models with high evidence $p(D|m)$ prefer both high accuracy and low complexity. Therefore, models with high evidence tend to predict the training data D well (high accuracy), yet also try to preserve the information encoded by the prior (low complexity). These types of models are said to *generalize* well, since they can be applied to different data sets without specific adaptations for each data set.

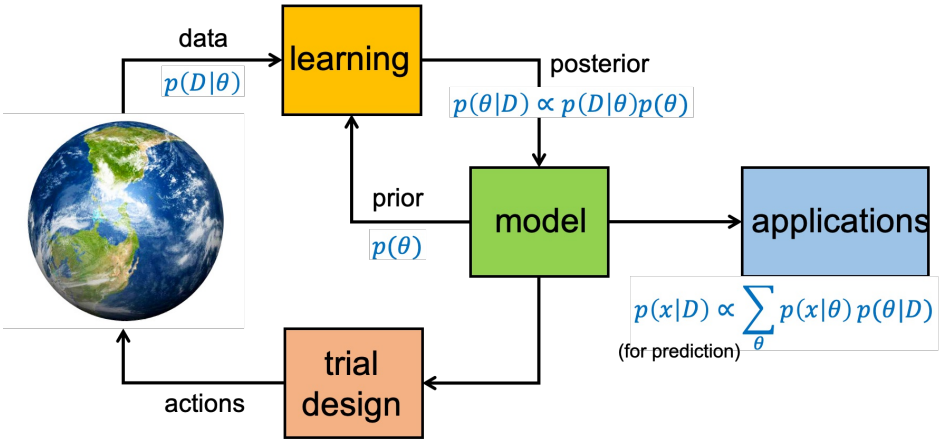
- Focussing only on accuracy maximization could lead to *overfitting*. Focussing only on complexity minimization could lead to *underfitting* of the model. Bayesian ML attends to both terms and avoids both underfitting and overfitting.

- $\Rightarrow$ Bayesian learning automatically leads to models that generalize well. There is **no need for early stopping or validation data sets**. There is also **no need for tuning parameters** in the learning process. Just learn on the full data set and all behaves well.

- This latter point accentuates that the common practice in machine learning to divide a data set into a training, test and validation set is just an ad hoc mechanism that compensates for failing to frame the learning task as a Bayesian inference task.

Bayesian Machine Learning and the Scientific Method Revisited

- The Bayesian design process provides a unified framework for the Scientific Inquiry method. We can now add equations to the design loop. (Trial design to be discussed in Intelligent Agent lesson.)



Revisiting the Challenge: Predicting a Coin Toss

At the beginning of this lesson, we posed the following challenge:

- We observe a the following sequence of heads (outcome = 1) and tails (outcome = 0) when tossing the same coin repeatedly

$$D = \{1011001\}\,.$$

- What is the probability that heads comes up next? We solve this in the next slides ...

Coin toss example (1): Model Specification

- We observe a sequence of N coin tosses $D = \{x_1, \ldots, x_N\}$ with n heads.

- Let us denote outcomes by

$$x_k = \begin{cases} 1 & \text{if heads comes up} \\ 0 & \text{otherwise (tails)} \end{cases}$$

Likelihood

- Assume a **Bernoulli** distributed variable $p(x_k = 1|\mu) = \mu$ for a single coin toss, leading to

$$p(x_k|\mu) = \mu^{x_k}(1-\mu)^{1-x_k}\,.$$

- Assume n times heads were thrown out of a total of N throws. The likelihood function then follows a **binomial** distribution :

$$p(D|\mu) = \prod_{k=1}^{N} p(x_k|\mu) = \mu^n(1-\mu)^{N-n}$$

Prior

- Assume the prior beliefs for μ are governed by a **beta distribution**

$$p(\mu) = \text{Beta}(\mu|\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\mu^{\alpha-1}(1-\mu)^{\beta-1}$$

where the Gamma function is sort-of a generalized factorial function. In particular, if α, β are integers, then

$$\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} = \frac{(\alpha+\beta-1)!}{(\alpha-1)!\,(\beta-1)!}$$

- A *what* distribution? Yes, the **beta distribution** is a **conjugate prior** for the binomial distribution, which means that

$$\underbrace{\text{beta}}_{\text{posterior}} \propto \underbrace{\text{binomial}}_{\text{likelihood}} \times \underbrace{\text{beta}}_{\text{prior}}$$

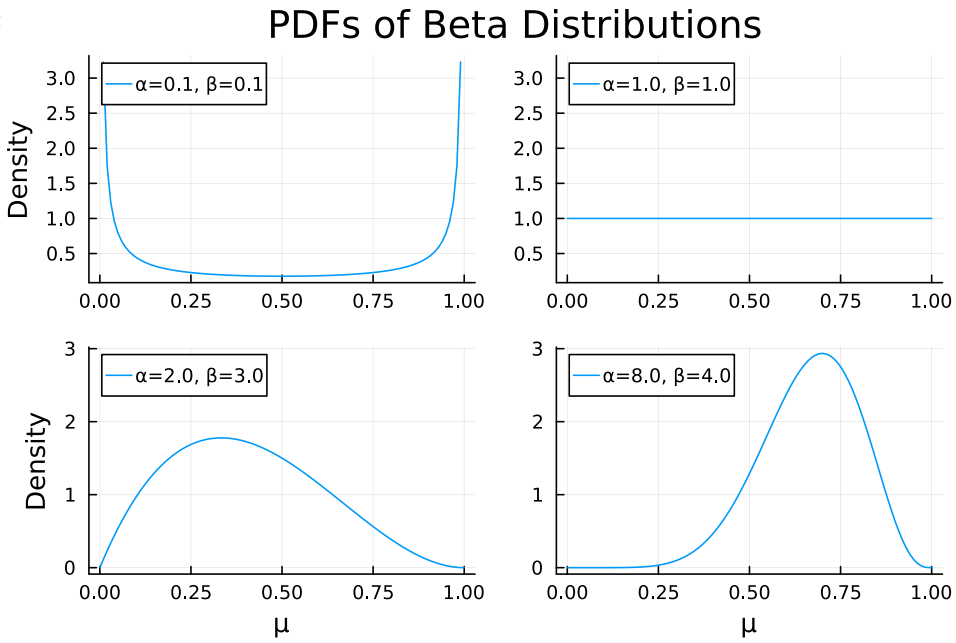
  so we get a closed-form posterior.

- α and β are called **hyperparameters**, since they parameterize the distribution for another parameter (μ). E.g., $\alpha = \beta = 1$ leads to a uniform prior for μ. We use Julia below to visualize some priors $\text{Beta}(\mu|\alpha,\beta)$ for different values for α, β.

In [1]:
```
using Pkg; Pkg.activate("../."); Pkg.instantiate();
using IJulia; try IJulia.clear_output(); catch _ end
using Distributions, StatsPlots, SpecialFunctions
using Plots, LaTeXStrings, Plots.PlotMeasures
```

In [2]:
```
# maintain a vector of log evidences to plot later
params = [
    (α=0.1, β=0.1),
    (α=1.0, β=1.0),
    (α=2.0, β=3.0),
    (α=8.0, β=4.0)
]

x = 0:0.01:1

plots = []
for (i, (α, β)) in enumerate(params)
    beta_dist = Beta(α, β)
    y = pdf.(beta_dist, x)

    xlabel = i in [3, 4] ? "μ" : ""
    ylabel = i in [1, 3] ? "Density" : ""

    push!(plots, plot(x, y, label="α=$α, β=$β", xlabel=xlabel, ylabel=ylabel))
end

plot(plots..., layout=(2, 2), suptitle="PDFs of Beta Distributions", legend=:topleft, link=:both, padding=10
```

Out[2]:



- Before observing any data, you can express your state-of-knowledge about the coin by choosing values for α and β that reflect your beliefs. Stronger yet, you *must* choose values for α and β, because the Bayesian framework does not allow you to walk away from your responsibility to explicitly state your beliefs before the experiment.

Coin toss example (2): Parameter estimation

- Infer posterior PDF over μ (and evidence) through Bayes rule

$$p(\mu|D) \cdot p(D) = p(D|\mu) \cdot p(\mu)$$

$$= \underbrace{\left(\mu^n (1-\mu)^{N-n}\right)}_{\text{likelihood}} \cdot \underbrace{\left(\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \mu^{\alpha-1}(1-\mu)^{\beta-1}\right)}_{\text{prior}}$$

$$= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \mu^{n+\alpha-1}(1-\mu)^{N-n+\beta-1}$$

$$= \underbrace{\left(\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \frac{\Gamma(n+\alpha)\Gamma(N-n+\beta)}{\Gamma(N+\alpha+\beta)}\right)}_{\text{evidence } p(D)} \cdot \underbrace{\left(\frac{\Gamma(N+\alpha+\beta)}{\Gamma(n+\alpha)\Gamma(N-n+\beta)} \mu^{n+\alpha-1}(1-\mu)^{N-n+\beta-1}\right)}_{\text{posterior } p(\mu|D)=\mathrm{Beta}(\mu|n+\alpha, N-n+\beta)}$$

hence the posterior is also beta-distributed as

$$p(\mu|D) = \mathrm{Beta}(\mu|\, n+\alpha, N-n+\beta)$$

Coin toss example (3): Model Evaluation

- It follow from the above calculation that the evidence for model m can be analytically expressed as

$$p(D|m) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \frac{\Gamma(n+\alpha)\Gamma(N-n+\beta)}{\Gamma(N+\alpha+\beta)}$$

- The model evidence is a scalar. The absolute value is not important. However, you may want to compare the model evidence of this model to the evidence for another model on the same data set.

Coin Toss Example (4): Prediction

- Once we have accepted a model, let's apply it to the application, in this case, predicting future observations.

- Marginalize over the parameter posterior to get the predictive PDF for a new coin toss $x_\bullet$, given the data D,

$$\begin{aligned} p(x_\bullet = 1|D) &= \int_0^1 p(x_\bullet = 1|\mu)\, p(\mu|D)\, \mathrm{d}\mu \\ &= \int_0^1 \mu \times \mathrm{Beta}(\mu|\, n+\alpha, N-n+\beta)\, \mathrm{d}\mu \\ &= \frac{n+\alpha}{N+\alpha+\beta} \end{aligned}$$

- This result is known as **Laplace's rule of succession**.

- The above integral computes the mean of a beta distribution, which is given by $\mathbb{E}[x] = \frac{a}{a+b}$ for $x \sim \mathrm{Beta}(a, b)$, see wikipedia.

- Finally, we're ready to solve our challenge: for $D = \{1011001\}$ and uniform prior ($\alpha = \beta = 1$), we get

$$p(x_\bullet = 1|D) = \frac{n+1}{N+2} = \frac{4+1}{7+2} = \frac{5}{9}$$

- In other words, given the model assumptions (the Bernoulli data-generating distribution and Beta prior as specified above), and the observations $D = \{1011001\}$, the probability for observing heads (outcome=1) on the next toss is $\frac{5}{9}$.

- Be aware that there is no such thing as an "objective" or "correct" prediction. Every prediction is conditional on the selected model and the used data set.

Coin Toss Example: What did we learn from the data?

- What did we learn from the data? Before seeing any data, we think that the probability for throwing heads is

$$\begin{aligned} p(x_\bullet = 1|D)|_{n=N=0} &= \left.\frac{n+\alpha}{N+\alpha+\beta}\right|_{n=N=0} \\ &= \frac{\alpha}{\alpha+\beta}\,. \end{aligned}$$

- Hence, α and β can be interpreted as prior pseudo-counts for heads and tails, respectively.

- If we were to assume zero pseudo-counts, i.e. $\alpha = \beta = 1$, then our prediction for throwing heads after N coin tosses is completely based on the data, given by

$$\begin{aligned} p(x_\bullet = 1|D)|_{\alpha=\beta=1} &= \left.\frac{n+\alpha}{N+\alpha+\beta}\right|_{\alpha=\beta=1} \\ &= \frac{n}{N}\,. \end{aligned}$$

- Note the following decomposition

$$\begin{aligned} p(x_\bullet = 1|\,D) &= \frac{n+\alpha}{N+\alpha+\beta} \\ &= \frac{\alpha}{N+\alpha+\beta} + \frac{n}{N+\alpha+\beta} \\ &= \frac{\alpha}{N+\alpha+\beta}\cdot\frac{\alpha+\beta}{\alpha+\beta} + \frac{n}{N+\alpha+\beta}\cdot\frac{N}{N} \\ &= \frac{\alpha}{\alpha+\beta}\cdot\frac{\alpha+\beta}{N+\alpha+\beta} + \frac{N}{N+\alpha+\beta}\cdot\frac{n}{N} \\ &= \frac{\alpha}{\alpha+\beta}\cdot\left(1-\frac{N}{N+\alpha+\beta}\right) + \frac{N}{N+\alpha+\beta} \cdot\frac{n}{N} \\ &= \underbrace{\frac{\alpha}{\alpha+\beta}}_{\substack{\text{prior}\\ \text{prediction}}} + \underbrace{\underbrace{\frac{N}{N+\alpha+\beta}}_{\text{gain}}\cdot\underbrace{\left(\underbrace{\frac{n}{N}}_{\substack{\text{data-based}\\ \text{prediction}}} - \underbrace{\frac{\alpha}{\alpha+\beta}}_{\substack{\text{prior}\\ \text{prediction}}}\right)}_{\text{prediction error}}}_{\text{correction}} \end{aligned}$$

- Let's interpret this decomposition of the posterior prediction. Before the data D was observed, our model generated a *prior prediction* $p(x_\bullet = 1) = \frac{\alpha}{\alpha+\beta}$. Next, the degree to which the actually observed data matches this prediction is represented by the *prediction error* $\frac{n}{N} - \frac{\alpha}{\alpha-\beta}$. The prior prediction is then updated to a *posterior prediction* $p(x_\bullet = 1|D)$ by adding a fraction of the prediction error to the prior prediction. Hence, the data plays the role of "correcting" the prior prediction.

- Note that, since $0 \leq \underbrace{\frac{N}{N+\alpha+\beta}}_{\text{gain}} < 1$, the Bayesian prediction lies between (fuses) the prior and data-based predictions.

- For large N, the gain goes to 1 and $p(x_\bullet = 1|D)|_{N\to\infty} \to \frac{n}{N}$ goes to the data-based prediction (the observed relative frequency).

Code Example: Bayesian evolution for the coin toss

- Next, we code an example for a sequence of coin tosses, where we assume that the true coin generates data $x_n \in \{0, 1\}$ by a Bernoulli distribution:

$$p(x_n|\mu = 0.4) = 0.4^{x_n} \cdot 0.6^{1-x_n}$$

- So, this coin is biased!
- In order predict the outcomes of future coin tosses, we'll compare two models.
- All models have the same data generating distribution (also Bernoulli)

$$p(x_n|\mu, m_k) = \mu^{x_n}(1-\mu)^{1-x_n} \quad \text{for } k = 1, 2$$

but they have different priors:

$$p(\mu|m_1) = \text{Beta}(\mu|\alpha = 100, \beta = 500)$$
$$p(\mu|m_2) = \text{Beta}(\mu|\alpha = 8, \beta = 13)$$

- You can verify that model m_2 has the best prior, since

$$p(x_n = 1|m_1) = \left.\frac{\alpha}{\alpha+\beta}\right|_{m_1} = 100/600 \approx 0.17$$
$$p(x_n = 1|m_2) = \left.\frac{\alpha}{\alpha+\beta}\right|_{m_2} = 8/21 \approx 0.38\,,$$

(but you are not supposed to know that the real coin has probability for heads $p(x_n = 1|\mu) = 0.4$).

- Let's run 500 tosses:

In [3]:
```
# computes log10 of Gamma function
function log10gamma(num)
    num = convert(BigInt, num)
    return log10(gamma(num))
end


μ  = 0.4;                          # specify model parameter
n_tosses = 500                     # specify number of coin tosses
samples = rand(n_tosses) .<= μ     # Flip 200 coins

function handle_coin_toss(prior :: Beta, observation :: Bool)
    posterior = Beta(prior.α + observation, prior.β + (1 - observation))
    return posterior
end

function log_evidence_prior(prior :: Beta, N :: Int64, n :: Int64)
    log_evidence = log10gamma(prior.α + prior.β) - log10gamma(prior.α) - log10gamma(prior.β) + log10gamma(n+
    return log_evidence
end

priors = [Beta(100., 500.), Beta(8., 13.)]  # specify prior distributions
n_models = length(priors)

# save a sequence of posterior distributions for every prior, starting with the prior itself
posterior_distributions = [[d] for d in priors]
log_evidences = [[] for _ in priors]

for (N, sample) in enumerate(samples)                          # for every sample we want to update our posteri
    for (i, prior) in enumerate(priors)                        # at every sample we want to update all distribu

        posterior = handle_coin_toss(prior, sample)            # do bayesian updating
        push!(posterior_distributions[i], posterior)           # add posterior to vector of posterior distribut
```

```
        # compute log evidence and add to vector
        log_evidence = log_evidence_prior(posterior_distributions[i][N], N, sum(samples[1:N]))
        push!(log_evidences[i], log_evidence)

        # the prior for the next sample is the posterior from the current sample
        priors[i] = posterior
    end
end
```

- For each model, as a function of the number of coin tosses, we plot the evolution of the parameter posteriors

$$p(\mu|D_n, m_\bullet)$$

In [4]:
```
#animate posterior distributions over time in a gif

anim = @animate for i in 1:n_tosses
    p = plot(title=string("n = ", i))
    for j in 1:n_models
        plot!(posterior_distributions[j][i+1], xlims = (0, 1), fill=(0, .2,), label=string("Posterior m", j)
    end
end

gif(anim, "figures/anim_bay_ct.gif", show_msg = false);
```

(This is an animation. If you are viewing these notes in PDF format you can see the animation at https://nbviewer.org/github/bertdv/BMLIP/blob/master/lessons/notebooks/Bayesian-Machine-Learning.ipynb .)

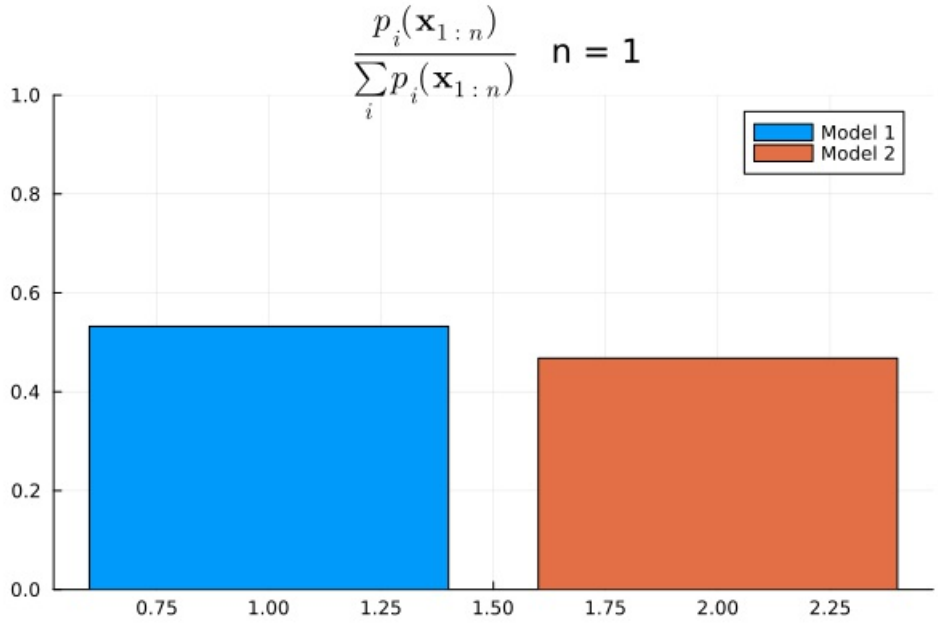
- Note that both posteriors move toward the "correct" value ($\mu = 0.4$). However, the posterior for m_1 (blue) moves much slower because we assumed far more pseudo-observations for m_1 than for m_2.
- As we get more observations, the influence of the prior diminishes.

- We have an intuition that m_2 is superior over m_1. Let's check this by plotting over time the relative Bayesian evidences for each model:

$$\frac{p(D_n|m_i)}{\sum_{i=1}^{2} p(D_n|m_i)}$$

In [5]:
```
using LaTeXStrings

evidences = map(model -> exp.(model), log_evidences)

anim = @animate for i in 1:n_tosses
    p = plot(title=string(L"\frac{p_i(\mathbf{x}_{1:n})}{\sum_i p_i(\mathbf{x}_{1:n})}","   n = ", i), ylims
    total = sum([evidences[j][i] for j in 1:n_models])
    bar!([(evidences[j][i] / total) for j in 1:n_models], group=["Model $i" for i in 1:n_models])
end

gif(anim, "figures/anim_bay_ct_log_evidence.gif", show_msg = false)
```

Out[5]:



- Over time, the relative evidence of model m_1 converges to 0. Can you explain this behavior?

From Posterior to Point-Estimate

- In the example above, Bayesian parameter estimation and prediction were tractable in closed-form. This is often not the case. We will need to approximate some of the computations.
- Recall Bayesian prediction

$$p(x|D) = \int p(x|\theta)p(\theta|D)\,\mathrm{d}\theta$$

- If we approximate posterior $p(\theta|D)$ by a delta function for one 'best' value $\hat{\theta}$, then the predictive distribution collapses to

$$p(x|D) = \int p(x|\theta)\,\delta(\theta - \hat{\theta})\,\mathrm{d}\theta = p(x|\hat{\theta})$$

- This is just the data generating distribution $p(x|\theta)$ evaluated at $\theta = \hat{\theta}$, which is easy to evaluate.
- The next question is how to get the parameter estimate $\hat{\theta}$? (See next slide).

Some Well-known Point-Estimates

- **Bayes estimate** (the mean of the posterior)

$$\hat{\theta}_{bayes} = \int \theta\, p\left(\theta|D\right)\,\mathrm{d}\theta$$

- **Maximum A Posteriori** (MAP) estimate

$$\hat{\theta}_{\mathrm{map}} = \arg\max_{\theta} p\left(\theta|D\right) = \arg\max_{\theta} p\left(D|\theta\right)$$
$$p\left(\theta\right)$$

- **Maximum Likelihood** (ML) estimate

$$\hat{\theta}_{ml} = \arg\max_{\theta} p\left(D|\theta\right)$$

  - Note that Maximum Likelihood is MAP with uniform prior
  - ML is the most common approximation to the full Bayesian posterior.

Bayesian vs Maximum Likelihood Learning

Consider the task: predict a datum x from an observed data set D.

	Bayesian	**Maximum Likelihood**
1. **Model Specification**	Choose a model m with data generating distribution $p(x\|\theta, m)$ and parameter prior $p(\theta\|m)$	Choose a model m with same data generating distribution $p(x\|\theta, m)$. No need for priors.
2. **Learning**	use Bayes rule to find the parameter posterior, $p(\theta\|D) \propto p(D\|\theta)p(\theta)$	By Maximum Likelihood (ML) optimization, $\hat{\theta} = \arg\max_{\theta} p(D\|\theta)$

3. **Prediction** $p(x|D) = \int p(x|\theta)p(\theta|D)\,\mathrm{d}\theta$ $p(x|D) = p(x|\hat{\theta})$

Report Card on Maximum Likelihood Estimation

- Maximum Likelihood (ML) is MAP with uniform prior. MAP is sometimes called a 'penalized' ML procedure:

$$\hat{\theta}_{map} = \arg\max_{\theta} \{\underbrace{\log p\left(D|\theta\right)}_{\text{log-likelihood}} + \underbrace{\log p\left(\theta\right)}_{\text{penalty}}\}$$

- (good!). ML works rather well if we have a lot of data because the influence of the prior diminishes with more data.
- (good!). Computationally often do-able. Useful fact that makes the optimization easier (since $\log$ is monotonously increasing):

$$\arg\max_{\theta} \log p(D|\theta) = \arg\max_{\theta} p(D|\theta)$$

- (bad). Cannot be used for model comparison! When doing ML estimation, the Bayesian model evidence always evaluates to zero because the prior probability mass under the likelihood function goes to zero. Therefore, when doing ML estimation, Bayesian model evidence cannot be used to evaluate model performance:

$$\begin{aligned}
\underbrace{p(D|m)}_{\substack{\text{Bayesian}\\ \text{evidence}}} &= \int p(D|\theta) \cdot p(\theta|m)\,\mathrm{d}\theta \\
&= \lim_{(b-a)\to\infty} \int p(D|\theta) \cdot \mathrm{Uniform}(\theta|a,b)\,\mathrm{d}\theta \\
&= \lim_{(b-a)\to\infty} \frac{1}{b-a} \underbrace{\int_a^b p(D|\theta)\,\mathrm{d}\theta}_{<\infty} \\
&= 0
\end{aligned}$$

  - In fact, this is a serious problem because Bayesian evidence is fundamentally the correct performance assessment criterion that follows from straighforward PT. In practice, when estimating parameters by maximum likelihood, we often evaluate model performance by an *ad hoc* performance measure such as mean-squared-error on a testing data set.

$\Rightarrow$ **Maximum Likelihood estimation is at best an approximation to Bayesian learning**, but for good reason a very popular learning method when faced with lots of available data.

OPTIONAL SLIDES

The Kullback-Leibler Divergence

- The Kullback-Leibler Divergence (a.k.a. relative entropy) between two distributions q and p is defined as

$$D_{\mathrm{KL}}[q,p] \equiv \sum_z q(z) \log \frac{q(z)}{p(z)}$$

- The following Gibbs Inequality holds (see wikipedia for proof):

$$D_{\mathrm{KL}}[q,p] \geq 0 \quad \text{with equality only if } p = q$$

- The KL divergence can be interpreted as a distance between two probability distributions.
- As an aside, note that $D_{\mathrm{KL}}[q,p] \neq D_{\mathrm{KL}}[p,q]$. Both divergences are relevant.
- Here is an animation that shows the KL divergence between two Gaussian distributions:

In [6]:
```
using Distributions, StatsPlots, Plots.PlotMeasures, LaTeXStrings

function kullback_leibler(q :: Normal, p :: Normal)
```

```
    # Calculates the KL Divergence between two gaussians
    # (see https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence for calculations)
    return log((p.σ / q.σ)) + ((q.σ)^2 + (p.μ - q.μ)^2) / (2*p.σ^2) - (1. / 2.)
end

# statistics of distributions we'll keep constant (we'll vary the mean of q)
# feel free to change these and see what happens
μ_p = 0
σ_p = 1
σ_q = 1

p = Normal(μ_p, σ_p)

anim = @animate for i in 1:100
    μ_seq = [(j / 10.) - 5. + μ_p for j in 1:i]
    kl = [kullback_leibler(Normal(μ, σ_q), p) for μ in μ_seq]
    viz = plot(right_margin=8mm, title=string(L"D_{KL}(Q || P) = ", round(100 * kl[i]) / 100.), legend=:topl
    μ_q = μ_seq[i]
    q = Normal(μ_q, σ_q)
    plot!(p, xlims = (μ_p - 8, μ_p + 8), fill=(0, .2,), label=string("P"), linewidth=2, ylims=(0,0.5))
    plot!(q, fill=(0, .2,), label=string("Q"), linewidth=2, ylims=(0,0.5))
    plot!(twinx(), μ_seq, kl, xticks=:none, ylims=(0, maximum(kl) + 3), linewidth = 3,
        legend=:topright,xlims = (μ_p - 8, μ_p + 8), color="green", label=L"D_{KL}(Q || P)")
end
gif(anim, "figures/anim_lat_kl.gif", show_msg = false);
```

- Note that this is an animation. If you are viewing these notes in PDF format you can see the animation at https://nbviewer.org/github/bertdv/BMLIP/blob/master/lessons/notebooks/Bayesian-Machine-Learning.ipynb

Factor Graphs

Preliminaries

- Goal
  - Introduction to Forney-style factor graphs and message passing-based inference
- Materials
  - Mandatory
    - These lecture notes
    - Loeliger (2007), The factor graph approach to model based signal processing, pp. 1295-1302 (until section V)
  - Optional
    - Frederico Wadehn (2015), Probabilistic graphical models: Factor graphs and more video lecture (**recommended**)
  - References
    - Forney (2001), Codes on graphs: normal realizations

Why Factor Graphs?

- A probabilistic inference task gets its computational load mainly through the need for marginalization (i.e., computing integrals). E.g., for a model $p(x_1, x_2, x_3, x_4, x_5)$, the inference task $p(x_2|x_3)$ is given by

$$p(x_2|x_3) = \frac{p(x_2, x_3)}{p(x_3)}$$
$$= \frac{\int \cdots \int p(x_1, x_2, x_3, x_4, x_5)\, \mathrm{d}x_1 \mathrm{d}x_4 \mathrm{d}x_5}{\int \cdots \int p(x_1, x_2, x_3, x_4, x_5)\, \mathrm{d}x_1 \mathrm{d}x_2 \mathrm{d}x_4 \mathrm{d}x_5}$$

- Since these computations (integrals or sums) suffer from the "curse of dimensionality", we often need to solve a simpler problem in order to get an answer.

- Factor graphs provide a computationally efficient approach to solving inference problems **if the probabilistic model can be factorized**.

- Factorization helps. For instance, if
  $p(x_1, x_2, x_3, x_4, x_5)$
  $= p(x_1)p(x_2, x_3)p(x_4)p(x_5|x_4)$
  , then

$$
\begin{aligned}
& p(x_2|x_3) \\
= & \frac{\int \cdots \int p(x_1)p(x_2,x_3)p(x_4)p(x_5|x_4)\,\mathrm{d}x_1\mathrm{d}x_4\mathrm{d}x_5}{\int \cdots \int p(x_1)p(x_2,x_3)p(x_4)p(x_5|x_4)} \\
& \mathrm{d}x_1\mathrm{d}x_2\mathrm{d}x_4\mathrm{d}x_5 \\
= & \frac{p(x_2,x_3)}{\int p(x_2,x_3)\mathrm{d}x_2}
\end{aligned}
$$

which is computationally much cheaper than the general case above.

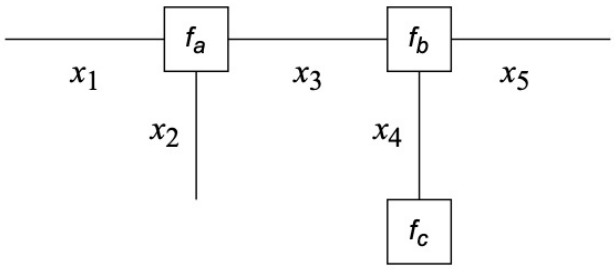
- In this lesson, we discuss how computationally efficient inference in *factorized* probability distributions can be automated by message passing-based inference in factor graphs.

Factor Graph Construction Rules

- Consider a function

$$
\begin{aligned}
f(x_1,x_2,x_3,x_4,x_5) = & f_a(x_1,x_2,x_3) \\
& \cdot f_b(x_3,x_4,x_5) \cdot f_c(x_4)
\end{aligned}
$$

- The factorization of this function can be graphically represented by a **Forney-style Factor Graph** (FFG):



- An FFG is an **undirected** graph subject to the following construction rules (Forney, 2001)

1. A **node** for every factor;
2. An **edge** (or **half-edge**) for every variable;
3. Node $f_\bullet$ is connected to edge x **iff** variable x appears in factor $f_\bullet$.

- A **configuration** is an assigment of values to all variables.
- A configuration $\omega = (x_1, x_2, x_3, x_4, x_5)$ is said to be **valid** iff $f(\omega) \neq 0$

Equality Nodes for Branching Points

- Note that a variable can appear in maximally two factors in an FFG (since an edge has only two end points).
- Consider the factorization (where x_2 appears in three factors)

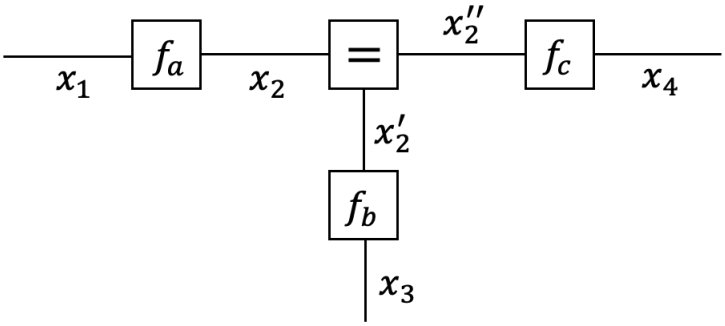
$$
\begin{aligned}
f(x_1,x_2,x_3,x_4) = & f_a(x_1,x_2) \cdot f_b(x_2,x_3) \\
& \cdot f_c(x_2,x_4)
\end{aligned}
$$

- For the factor graph representation, we will instead consider the function g, defined as

$$
\begin{aligned}
g(x_1,x_2 & ,x_2',x_2'',x_3,x_4) = f_a(x_1,x_2) \cdot f_b(x_2',x_3) \\
& \cdot f_c(x_2'',x_4) \cdot f_=(x_2,x_2',x_2'')
\end{aligned}
$$

where

$$
f_=(x_2,x_2',x_2'') \triangleq \delta(x_2-x_2')\,\delta(x_2-x_2'')
$$



- Note that through introduction of auxiliary variables X_2' and X_2'' and a factor $f_=(x_2, x_2', x_2'')$, each variable in g appears in maximally two factors.
- The constraint $f_=(x, x', x'')$ enforces that $X = X' = X''$ **for every valid configuration**.
- Since f is a marginal of g, i.e.,

$$f(x_1, x_2, x_3, x_4) = \iint g(x_1, x_2, x_2', x_2'', x_3, x_4)\,\mathrm{d}x_2'\mathrm{d}x_2''$$

  it follows that any inference problem on f can be executed by a corresponding inference problem on g, e.g.,

$$\begin{aligned} f(x_1 \mid x_2) &\triangleq \frac{\iint f(x_1, x_2, x_3, x_4)\,\mathrm{d}x_3\mathrm{d}x_4}{\int\cdots\int f(x_1, x_2, x_3, x_4)\,\mathrm{d}x_1\mathrm{d}x_3\mathrm{d}x_4} \\ &= \frac{\int\cdots\int g(x_1, x_2, x_2', x_2'', x_3, x_4)\,\mathrm{d}x_2'\mathrm{d}x_2''\mathrm{d}x_3\mathrm{d}x_4}{\int\cdots\int g(x_1, x_2, x_2', x_2'', x_3, x_4)\,\mathrm{d}x_1\mathrm{d}x_2'\mathrm{d}x_2''\mathrm{d}x_3\mathrm{d}x_4} \\ &= g(x_1 \mid x_2) \end{aligned}$$

- $\Rightarrow$ **Any factorization of a global function f can be represented by a Forney-style Factor Graph**.

Probabilistic Models as Factor Graphs

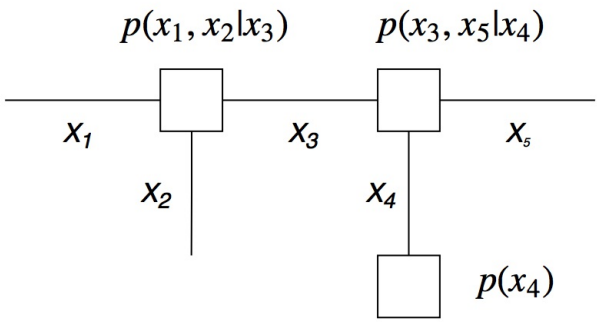
- FFGs can be used to express conditional independence (factorization) in probabilistic models.
- For example, the (previously shown) graph for $f_a(x_1, x_2, x_3) \cdot f_b(x_3, x_4, x_5) \cdot f_c(x_4)$ could represent the probabilistic model

$$p(x_1, x_2, x_3, x_4, x_5) = p(x_1, x_2|x_3) \cdot p(x_3, x_5|x_4) \cdot p(x_4)$$

  where we identify

$$\begin{aligned} f_a(x_1, x_2, x_3) &= p(x_1, x_2|x_3) \\ f_b(x_3, x_4, x_5) &= p(x_3, x_5|x_4) \\ f_c(x_4) &= p(x_4) \end{aligned}$$

- This is the graph



Inference by Closing Boxes

- Factorizations provide opportunities to cut on the amount of needed computations when doing inference. In what follows, we will use FFGs to process these opportunities in an automatic way by message passing between the nodes of the graph.

- Assume we wish to compute the marginal

$$\bar{f}(x_3) \triangleq \sum_{x_1,x_2,x_4,x_5,x_6,x_7} f(x_1, x_2, \ldots, x_7)$$

  for a model f with given factorization

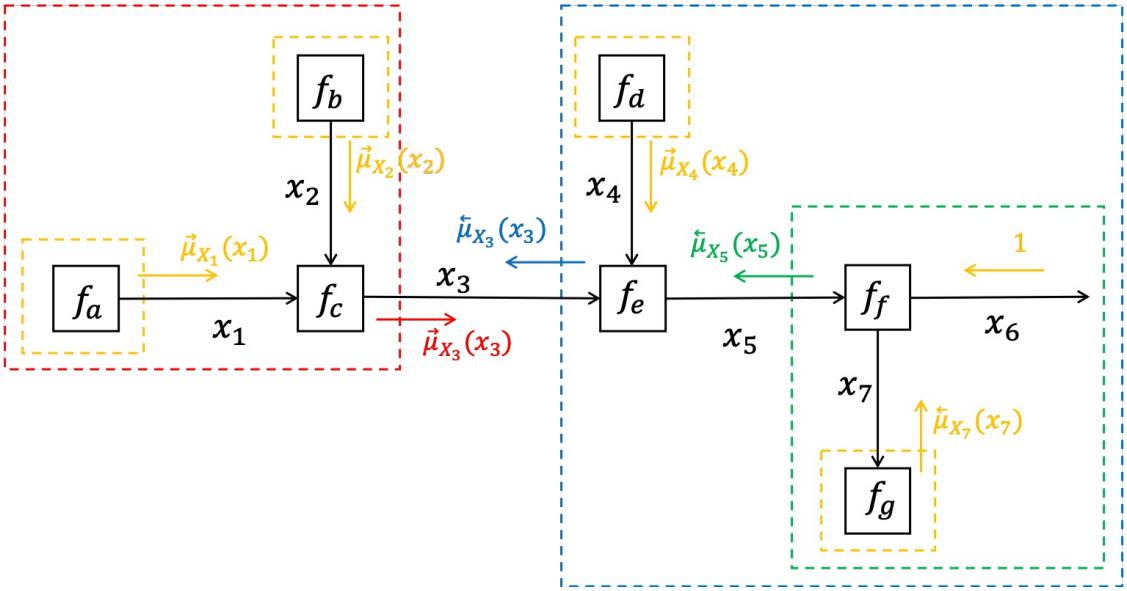
$$f(x_1, x_2, \ldots, x_7) = f_a(x_1) f_b(x_2) f_c(x_1, x_2, x_3) f_d(x_4) f_e(x_3, x_4, x_5) f_f(x_5, x_6, x_7) f_g(x_7)$$

- Note that, if each variable x_i can take on 10 values, then the computing the marginal $\bar{f}(x_3)$ takes about 10^6 (1 million) additions.

- Due to the factorization and the Generalized Distributive Law, we can decompose this sum-of-products to the following product-of-sums:

$$\bar{f}(x_3) = \underbrace{\left(\sum_{x_1,x_2} \underbrace{f_a(x_1)}_{\vec{\mu}_{X_1}(x_1)} \underbrace{f_b(x_2)}_{\vec{\mu}_{X_2}(x_2)} f_c(x_1, x_2, x_3) \right)}_{\vec{\mu}_{X_3}(x_3)} \cdot \underbrace{\left(\sum_{x_4,x_5} \underbrace{f_d(x_4)}_{\vec{\mu}_{X_4}(x_4)} f_e(x_3, x_4, x_5) \cdot \underbrace{\underbrace{\left(\sum_{x_6,x_7} f_f(x_5, x_6, x_7) \underbrace{f_g(x_7)}_{\overleftarrow{\mu}_{X_7}(x_7)} \right)}_{\overleftarrow{\mu}_{X_5}(x_5)}}_{\overleftarrow{\mu}_{X_3}(x_3)} \right)}$$

  which, in case x_i has 10 values, requires a few hundred additions and is therefore computationally (much!) lighter than executing the full sum $\sum_{x_1,\ldots,x_7} f(x_1, x_2, \ldots, x_7)$

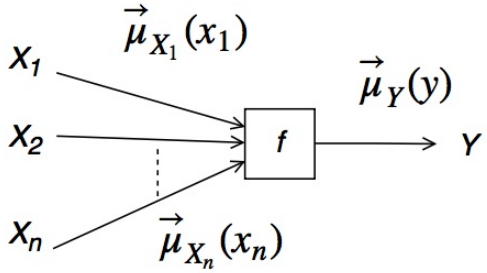


- Note that the auxiliary factor $\overrightarrow{\mu}_{X_3}(x_3)$ is obtained by multiplying all enclosed factors (f_a, f_b, f_c) by the red dashed box, followed by marginalization (summing) over all enclosed variables (x_1, x_2).
- This is the **Closing the Box**-rule, which is a general recipe for marginalization of latent variables (inside the box) and leads to a new factor that has the variables (edges) that cross the box as arguments. For instance, the argument of the remaining factor $\overrightarrow{\mu}_{X_3}(x_3)$ is the variable on the edge that crosses the red box (x_3).
- Hence, $\overrightarrow{\mu}_{X_3}(x_3)$ can be interpreted as a **message from the red box toward variable** x_3.
- We drew *directed edges* in the FFG in order to distinguish forward messages $\overrightarrow{\mu}_{\bullet}(\cdot)$ (in the same direction as the arrow of the edge) from backward messages $\overleftarrow{\mu}_{\bullet}(\cdot)$ (in opposite direction). This is just a notational convenience since an FFG is computationally an undirected graph.

Sum-Product Algorithm

- Closing-the-box can also be interpreted as a **message update rule** for an outgoing message from a node. For a node $f(y, x_1, \ldots, x_n)$ with incoming messages $\overrightarrow{\mu}_{X_1}(x_1)$, $\overrightarrow{\mu}_{X_1}(x_1), \ldots, \overrightarrow{\mu}_{X_n}(x_n)$, the outgoing message is given by (Loeliger (2007), pg.1299):

$$\underbrace{\overrightarrow{\mu}_{Y}(y)}_{\substack{\text{outgoing}\\ \text{message}}} = \sum_{x_1,\ldots,x_n} \underbrace{\overrightarrow{\mu}_{X_1}(x_1)\cdots\overrightarrow{\mu}_{X_n}(x_n)}_{\substack{\text{incoming}\\ \text{messages}}} \cdot \underbrace{f(y, x_1, \ldots, x_n)}_{\substack{\text{node}\\ \text{function}}}$$



- This is called the **Sum-Product Message** (SPM) update rule. (Look at the formula to understand why it's called the SPM update rule).
- Note that all SPM update rules can be computed from information that is **locally available** at each node.

- If the factor graph for a function f has **no cycles** (i.e., the graph is a tree), then the marginal $\bar{f}(x_3) = \sum_{x_1,x_2,x_4,x_5,x_6,x_7} f(x_1, x_2, \ldots, x_7)$ is given by multiplying the forward and backward messages on that edge:

$$\boxed{\bar{f}(x_3) = \overrightarrow{\mu}_{X_3}(x_3) \cdot \overleftarrow{\mu}_{X_3}(x_3)}$$

- It follows that the marginal $\bar{f}(x_3) = \sum_{x_1,x_2,x_4,x_5,x_6,x_7} f(x_1, x_2, \ldots, x_7)$ can be efficiently computed through sum-product messages. Executing inference through SP message passing is called the **Sum-Product Algorithm** (or alternatively, the **belief propagation** algorithm).

- Just as a final note, inference by sum-product message passing is much like replacing the sum-of-products

$$ac + ad + bc + bd$$

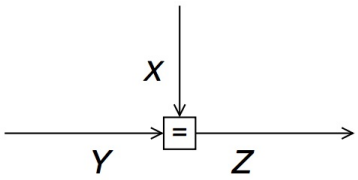
by the following product-of-sums:

$$(a + b)(c + d)\,.$$

- Which of these two computations is cheaper to execute?

Sum-Product Messages for the Equality Node

- As an example, let´s evaluate the SP messages for the **equality node** $f_=(x, y, z) = \delta(z - x)\delta(z - y)$:



$$\begin{aligned}
\overrightarrow{\mu}_Z(z) &= \iint \overrightarrow{\mu}_X(x)\overrightarrow{\mu}_Y(y)\,\delta(z - x)\delta(z - y)\,\mathrm{d}x\mathrm{d}y \\
&= \overrightarrow{\mu}_X(z) \int \overrightarrow{\mu}_Y(y)\,\delta(z - y)\,\mathrm{d}y \\
&= \overrightarrow{\mu}_X(z)\overrightarrow{\mu}_Y(z)
\end{aligned}$$

- By symmetry, this also implies (for the same equality node) that

$$\begin{aligned}
\overleftarrow{\mu}_X(x) &= \overrightarrow{\mu}_Y(x)\overleftarrow{\mu}_Z(x) \quad \text{and} \\
\overleftarrow{\mu}_Y(y) &= \overrightarrow{\mu}_X(y)\overleftarrow{\mu}_Z(y)\,.
\end{aligned}$$

- Let us now consider the case of Gaussian messages $\overrightarrow{\mu}_X(x) = \mathcal{N}(x|\overrightarrow{m}_X, \overrightarrow{V}_X)$, $\overrightarrow{\mu}_Y(y) = \mathcal{N}(y|\overrightarrow{m}_Y, \overrightarrow{V}_Y)$ and $\overrightarrow{\mu}_Z(z) = \mathcal{N}(z|\overrightarrow{m}_Z, \overrightarrow{V}_Z)$. Let´s also define the precision matrices $\overrightarrow{W}_X \triangleq \overrightarrow{V}_X^{-1}$ and similarly for Y and Z. Then applying the SP update rule leads to multiplication of two Gaussian distributions (see Roweis notes), resulting in

$$\begin{aligned}
\overrightarrow{W}_Z &= \overrightarrow{W}_X + \overrightarrow{W}_Y && \text{(precisions add)} \\
\overrightarrow{W}_Z\overrightarrow{m}_z &= \overrightarrow{W}_X\overrightarrow{m}_X + \overrightarrow{W}_Y\overrightarrow{m}_Y && \text{(natural means add)}
\end{aligned}$$

- It follows that **message passing through an equality node is similar to applying Bayes rule**, i.e., fusion of two information sources. Does this make sense?

Message Passing Schedules

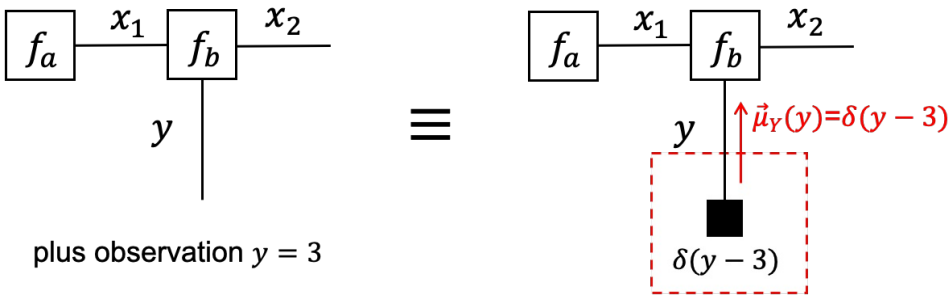
- In a non-cyclic (ie, tree) graph, start with messages from the terminals and keep passing messages through the internal nodes towards the "target" variable (x_3 in above problem) until you have both the forward and backward message for the target variable.

- In a tree graph, if you continue to pass messages throughout the graph, the Sum-Product Algorithm computes **exact** marginals for all hidden variables.

- If the graph contains cycles, we have in principle an infinite tree by "unrolling" the graph. In this case, the SP Algorithm is not guaranteed to find exact marginals. In practice, if we apply the SP algorithm for just a few iterations ("unrolls"), then we often find satisfying approximate marginals.

Terminal Nodes and Processing Observations

- We can use terminal nodes to represent observations, e.g., add a factor $f(y) = \delta(y-3)$ to terminate the half-edge for variable Y if $y = 3$ is observed.

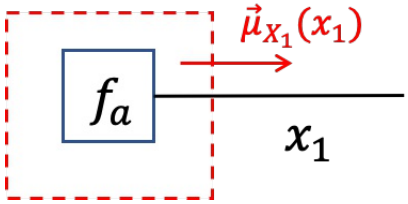


- Terminal nodes that carry observations are denoted by small black boxes.

- The message out of a **terminal node** (attached to only 1 edge) is the factor itself. For instance, closing a box around terminal node $f_a(x_1)$ would lead to

$$\overrightarrow{\mu}_{X_1}(x_1) \triangleq \sum_{\substack{\text{enclosed} \\ \text{variables}}} \prod_{\substack{\text{enclosed} \\ \text{factors}}} f_a(x_1) = f_a(x_1)$$

  since there are no enclosed variables.



- The message from a half-edge is 1 (one). You can verify this by imagining that a half-edge x can be terminated by a node function $f(x) = 1$ without affecting any inference issue.

Automating Bayesian Inference by Message Passing

- The foregoing message update rules can be worked out in closed-form and put into tables (e.g., see Tables 1 through 6 in Loeliger (2007) for many standard factors such as essential probability distributions and operations such as additions, fixed-gain multiplications and branching (equality nodes).
- In the optional slides below, we have worked out a few more update rules for the addition node and the multiplication node.
- If the update rules for all node types in a graph have been tabulated, then inference by message passing comes down to executing a set of table-lookup operations, thus creating a completely **automatable Bayesian inference framework**.
- In our research lab BIASlab (FLUX 7.060), we are developing RxInfer, which is a (Julia) toolbox for automating Bayesian inference by message passing in a factor graph.

Example: Bayesian Linear Regression by Message Passing

- Assume we want to estimate some function $f : \mathbb{R}^D \rightarrow \mathbb{R}$ from a given data set $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, with model assumption $y_i = f(x_i) + \epsilon_i$.

model specification

- We will assume a linear model with white Gaussian noise and a Gaussian prior on the coefficients w:

$$y_i = w^T x_i + \epsilon_i$$
$$\epsilon_i \sim \mathcal{N}(0, \sigma^2)$$
$$w \sim \mathcal{N}(0, \Sigma)$$

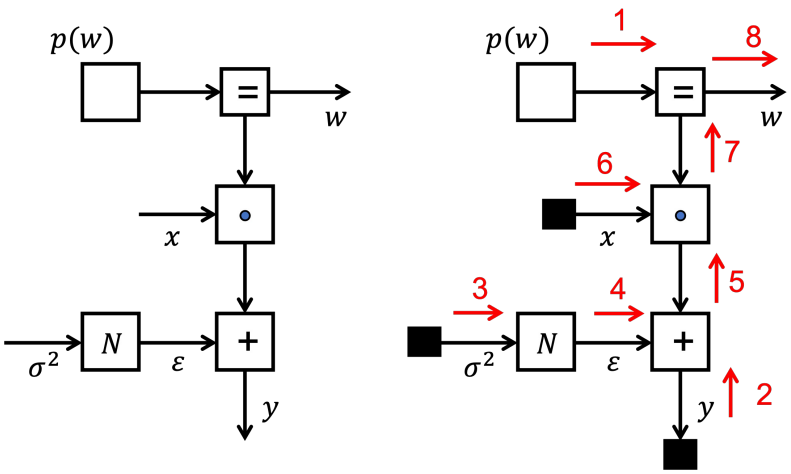
or equivalently

$$p(w, \epsilon, D) = \overbrace{p(w)}^{\text{weight prior}} \prod_{i=1}^{N} \overbrace{p(y_i \mid x_i, w, \epsilon_i)}^{\text{regression model}} \overbrace{p(\epsilon_i)}^{\text{noise model}}$$
$$= \mathcal{N}(w \mid 0, \Sigma) \prod_{i=1}^{N} \delta(y_i - w^T x_i - \epsilon_i) \mathcal{N}(\epsilon_i \mid 0, \sigma^2)$$

Inference (parameter estimation)

- We are interested in inferring the posterior $p(w|D)$. We will execute inference by message passing on the FFG for the model.

- The left figure shows the factor graph for this model.
- The right figure shows the message passing scheme.



CODE EXAMPLE

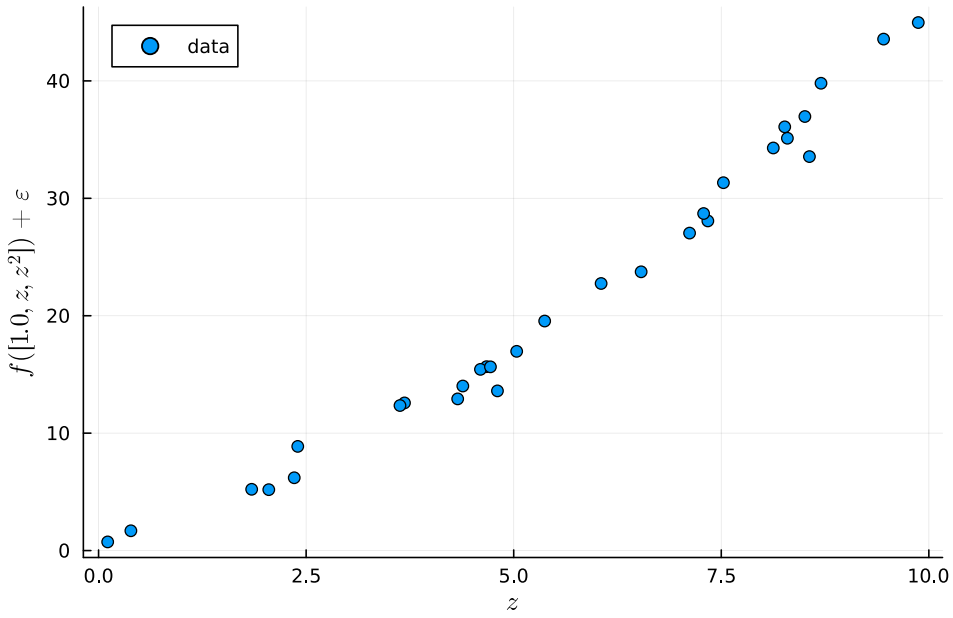
Let's solve this problem by message passing-based inference with Julia's FFG toolbox RxInfer.

In [1]:
```
using Pkg; Pkg.activate("../."); Pkg.instantiate();
using IJulia; try IJulia.clear_output(); catch _ end
```

Out[1]:0

In [2]:
```
using Plots, LinearAlgebra, LaTeXStrings

# Parameters
Σ = 1e5 * Diagonal(I,3) # Covariance matrix of prior on w
σ2 = 2.0                # Noise variance

# Generate data set
w = [1.0; 2.0; 0.25]
N = 30
z = 10.0*rand(N)
x_train = [[1.0; z; z^2] for z in z] # Feature vector x = [1.0; z; z^2]
f(x) = (w'*x)[1]
y_train = map(f, x_train) + sqrt(σ2)*randn(N) # y[i] = w' * x[i] + ϵ
scatter(z, y_train, label="data", xlabel=L"z", ylabel=L"f([1.0, z, z^2]) + \epsilon")
```

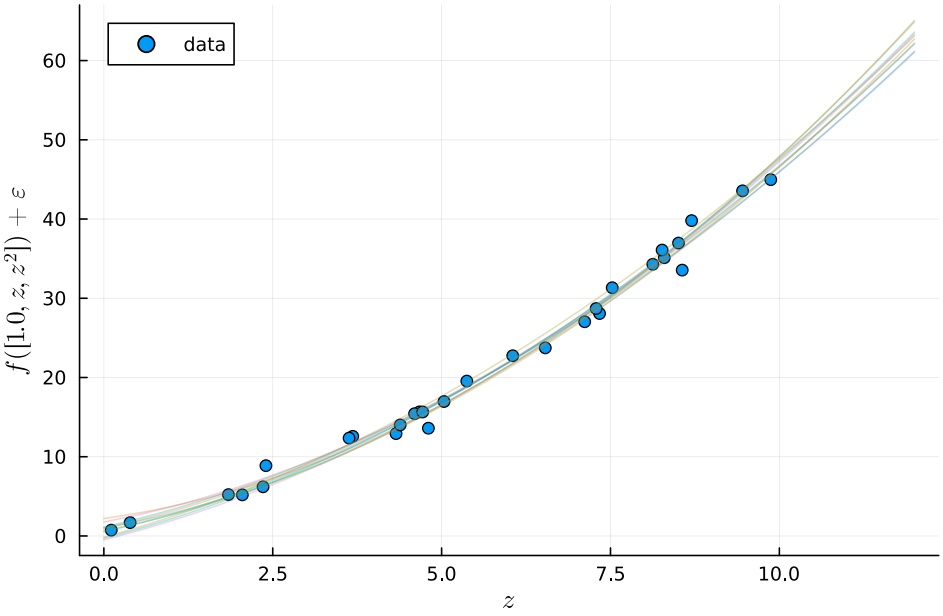
Out[2]:



Now build the factor graph in RxInfer, perform sum-product message passing and plot results (mean of posterior).

In [3]:
```
using RxInfer, Random
# Build model
@model function linear_regression(y,x, N, Σ, σ2)

    w ~ MvNormalMeanCovariance(zeros(3),Σ)

    for i in 1:N
        y[i] ~ NormalMeanVariance(dot(w , x[i]), σ2)
    end
end
# Run message passing algorithm
results = infer(
    model      = linear_regression(N=length(x_train), Σ=Σ, σ2=σ2),
    data       = (y = y_train, x = x_train),
    returnvars = (w = KeepLast(),),
    iterations = 20,
);
# Plot result
w = results.posteriors[:w]
println("Posterior distribution of w: $(w)")
plt = scatter(z, y_train, label="data", xlabel=L"z", ylabel=L"f([1.0, z, z^2]) + \epsilon")
z_test = collect(0:0.2:12)
x_test = [[1.0; z; z^2] for z in z_test]
for i=1:10
    w_sample = rand(results.posteriors[:w])
    f_est(x) = (w_sample'*x)[1]
    plt = plot!(z_test, map(f_est, x_test), alpha=0.3, label="");
end
display(plt)
```

```
Posterior distribution of w: MvNormalWeightedMeanPrecision(
xi: [321.2944016185268, 2273.792327424786, 17447.3912508887]
Λ: [15.00001 83.01124127862506 565.6888727430213; 83.01124127862506 565.6888827430213 4227.405944116379; 565.688
8727430213 4227.405944116379 33430.372100648296]
)
```



Final thoughts: Modularity and Abstraction

- The great Michael Jordan (no, not this one, but this one), wrote:

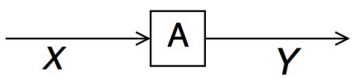
  > "I basically know of two principles for treating complicated systems in simple ways: the first is the principle of **modularity** and the second is the principle of **abstraction**. I am an apologist for computational probability in machine learning because I believe that probability theory implements these two principles in deep and intriguing ways — namely through factorization and through averaging. Exploiting these two mechanisms as fully as possible seems to me to be the way forward in machine learning." — Michael Jordan, 1997 (quoted in Fre98).

- Factor graphs realize these ideas nicely, both visually and computationally.
- Visually, the modularity of conditional independencies in the model are displayed by the graph structure. Each node hides internal complexity and by closing-the-box, we can hierarchically move on to higher levels of abstraction.
- Computationally, message passing-based inference uses the Distributive Law to avoid any unnecessary computations.
- What is the relevance of this lesson? RxInfer is not yet a finished project. Still, my prediction is that in 5-10 years, this lesson on Factor Graphs will be the final lecture of part-A of this class, aimed at engineers who need to develop machine learning applications. In principle you have all the tools now to work out the 4-step machine learning recipe (1. model specification, 2. parameter learning, 3. model evaluation, 4. application) that was proposed in the Bayesian machine learning lesson. You can propose any model and execute the (learning, evaluation, and application) stages by executing the corresponding inference task automatically in RxInfer.
- Part-B of this class would be about on advanced methods on how to improve automated inference by RxInfer or a similar probabilistic programming package. The Bayesian approach fully supports separating model specification from the inference task.

OPTIONAL SLIDES

Sum-Product Messages for Multiplication Nodes

- Next, let us consider a **multiplication** by a fixed (invertible matrix) gain $f_A(x, y) = \delta(y - Ax)$



$$\begin{aligned}\overrightarrow{\mu}_Y(y) &= \int \overrightarrow{\mu}_X(x)\, \delta(y - Ax)\, \mathrm{d}x \\ &= \int \overrightarrow{\mu}_X(x)\, |A|^{-1} \delta(x - A^{-1}y)\, \mathrm{d}x \\ &= |A|^{-1} \overrightarrow{\mu}_X(A^{-1}y)\,.\end{aligned}$$

- For a Gaussian message input message $\overrightarrow{\mu}_X(x) = \mathcal{N}(x|\overrightarrow{m}_X, \overrightarrow{V}_X)$, the output message is also Gaussian with

$$\overrightarrow{m}_Y = A\overrightarrow{m}_X \text{, and } \overrightarrow{V}_Y = A\overrightarrow{V}_X A^T$$

since

$$\begin{aligned}\overrightarrow{\mu}_Y(y) &= |A|^{-1}\overrightarrow{\mu}_X(A^{-1}y) \\ &\propto \exp\left(-\frac{1}{2}\left(A^{-1}y - \overrightarrow{m}_X\right)^T \overrightarrow{V}_X^{-1}\left(A^{-1}y - \overrightarrow{m}_X\right)\right) \\ &= \exp\Big(-\frac{1}{2}\left(y - A\overrightarrow{m}_X\right)^T \underbrace{A^{-T}\overrightarrow{V}_X^{-1}A^{-1}}_{(A\overrightarrow{V}_X A^T)^{-1}}\left(y - A\overrightarrow{m}_X\right)\Big) \\ &\propto \mathcal{N}(y|A\overrightarrow{m}_X, A\overrightarrow{V}_X A^T)\,.\end{aligned}$$

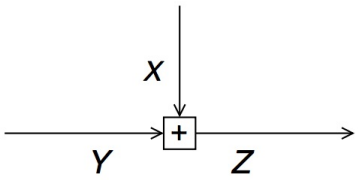
- **Exercise**: Proof that, for the same factor $\delta(y - Ax)$ and Gaussian messages, the (backward) sum-product message $\overleftarrow{\mu}_X$ is given by

$$\begin{aligned}\overleftarrow{\xi}_X &= A^T\overleftarrow{\xi}_Y \\ \overleftarrow{W}_X &= A^T\overleftarrow{W}_Y A\end{aligned}$$

where $\overleftarrow{\xi}_X \triangleq \overleftarrow{W}_X\overleftarrow{m}_X$ and $\overleftarrow{W}_X \triangleq \overleftarrow{V}_X^{-1}$ (and similarly for Y).

Code example: Gaussian forward and backward messages for the Addition node

Let's calculate the Gaussian forward and backward messages for the addition node in RxInfer.



In [4]:
```
println("Forward message on Z:")
@call_rule typeof(+)(:out, Marginalisation) (m_in1 = NormalMeanVariance(1.0, 1.0), m_in2 = NormalMeanVarianc
```

```
Forward message on Z:
```

Out[4]:
```
NormalMeanVariance{Float64}(μ=3.0, v=2.0)
```

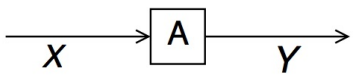
In [5]:
```
println("Backward message on X:")
@call_rule typeof(+)(:in1, Marginalisation) (m_out = NormalMeanVariance(3.0, 1.0), m_in2 = NormalMeanVarianc
```

```
Backward message on X:
```

Out[5]:
```
NormalMeanVariance{Float64}(μ=1.0, v=2.0)
```

Code Example: forward and backward messages for the Matrix Multiplication node

In the same way we can also investigate the forward and backward messages for the matrix multiplication ("gain") node



In [6]:
```
println("Forward message on Y:")
@call_rule typeof(*)(:out, Marginalisation) (m_A = PointMass(4.0), m_in = NormalMeanVariance(1.0, 1.0))
```

```
Forward message on Y:
```

Out[6]:
```
NormalMeanVariance{Float64}(μ=4.0, v=16.0)
```

In [7]:
```
println("Backward message on X:")
```

```
@call_rule typeof(*)(:in, Marginalisation) (m_out = NormalMeanVariance(2.0, 1.0), m_A = PointMass(4.0))
```

```
Backward message on X:
```

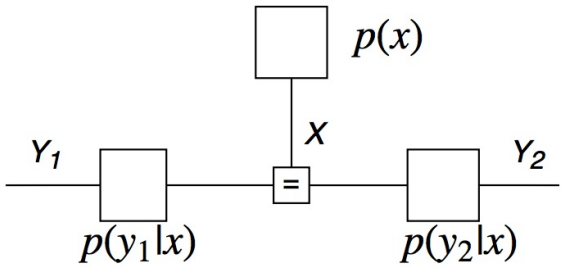
Out[7]:`NormalWeightedMeanPrecision{Float64}(xi=8.0, w=16.0)`

Example: Sum-Product Algorithm to infer a posterior

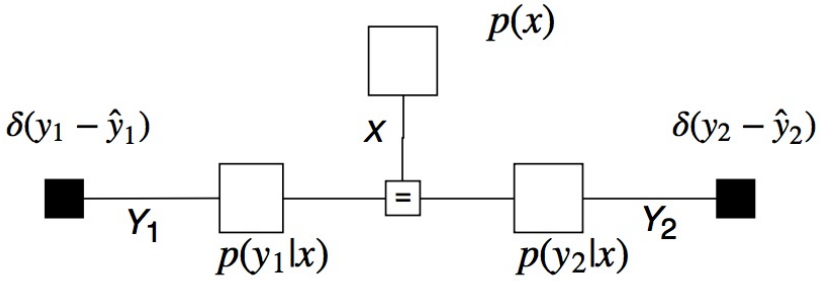
- Consider a generative model

$$p(x, y_1, y_2) = p(x)\, p(y_1|x)\, p(y_2|x).$$

  - This model expresses the assumption that Y_1 and Y_2 are independent measurements of X.



- Assume that we are interested in the posterior for X after observing $Y_1 = \hat{y}_1$ and $Y_2 = \hat{y}_2$. The posterior for X can be inferred by applying the sum-product algorithm to the following graph:



- (Note that) we usually draw terminal nodes for observed variables in the graph by smaller solid-black squares. This is just to help the visualization of the graph, since the computational rules are no different than for other nodes.

Code for Sum-Product Algorithm to infer a posterior

We'll use RxInfer to build the above graph, and perform sum-product message passing to infer the posterior $p(x|y_1, y_2)$. We assume $p(y_1|x)$ and $p(y_2|x)$ to be Gaussian likelihoods with known variances:

$$\begin{aligned} p(y_1 \mid x) &= \mathcal{N}(y_1 \mid x, v_{y1}) \\ p(y_2 \mid x) &= \mathcal{N}(y_2 \mid x, v_{y2}) \end{aligned}$$

Under this model, the posterior is given by:

$$\begin{aligned} p(x \mid y_1, y_2) &\propto \overbrace{p(y_1 \mid x)\, p(y_2 \mid x)}^{\text{likelihood}} \overbrace{p(x)}^{\text{prior}} \\ &= \mathcal{N}(x \mid \hat{y}_1, v_{y1})\, \mathcal{N}(x \mid \hat{y}_2, v_{y2})\, \mathcal{N}(x \mid m_x, v_x) \end{aligned}$$

so we can validate the answer by solving the Gaussian multiplication manually.

In [8]:
```
# Data
y1_hat = 1.0
y2_hat = 2.0

# Construct the factor graph
@model function my_model(y1,y2)

    # `x` is the hidden states
    x ~ NormalMeanVariance(0.0, 4.0)
```

```
        # `y1` and `y2` are "clamped" observations
        y1 ~ NormalMeanVariance(x, 1.0)
        y2 ~ NormalMeanVariance(x, 2.0)

        return x
    end

    result = infer(model=my_model(), data=(y1=y1_hat, y2 = y2_hat,))
    println("Sum-product message passing result: p(x|y1,y2) = 𝒩($(mean(result.posteriors[:x])),$(var(result.pos

    # Calculate mean and variance of p(x|y1,y2) manually by multiplying 3 Gaussians (see lesson 4 for details)
    v = 1 / (1/4 + 1/1 + 1/2)
    m = v * (0/4 + y1_hat/1.0 + y2_hat/2.0)
    println("Manual result: p(x|y1,y2) = 𝒩($(m), $(v))")
```

```
Sum-product message passing result: p(x|y1,y2) = 𝒩(1.1428571428571428,0.5714285714285714)
Manual result: p(x|y1,y2) = 𝒩(1.1428571428571428, 0.5714285714285714)
```

Continuous Data and the Gaussian Distribution

Preliminaries

- Goal
  - Review of information processing with Gaussian distributions in linear systems
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 85-93
    - MacKay - 2006 - The Humble Gaussian Distribution (highly recommended!)
    - Ariel Caticha - 2012 - Entropic Inference and the Foundations of Physics, pp.30-34, section 2.8, the Gaussian distribution
  - References
    - E.T. Jaynes - 2003 - Probability Theory, The Logic of Science (best book available on the Bayesian view on probability theory)

Example Problem

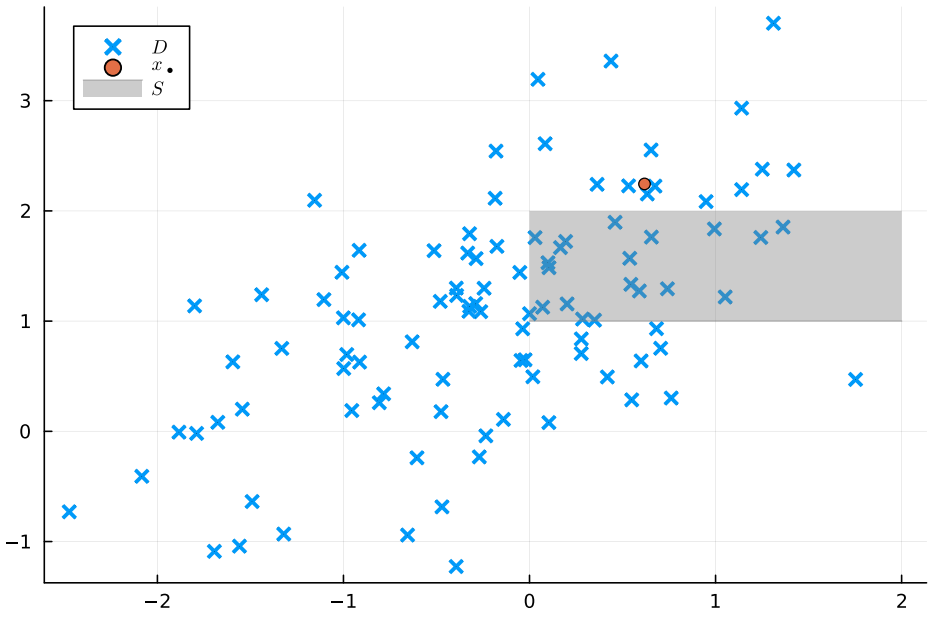
Consider a set of observations $D = \{x_1, \ldots, x_N\}$ in the 2-dimensional plane (see Figure). All observations were generated by the same process. We now draw an extra observation $x_\bullet = (a, b)$ from the same data generating process. What is the probability that $x_\bullet$ lies within the shaded rectangle S?

In [1]:
```
using Pkg; Pkg.activate("../."); Pkg.instantiate();
using IJulia; try IJulia.clear_output(); catch _ end
```

Out[1]:0

In [2]:
```
using Distributions, Plots, LaTeXStrings

N = 100
generative_dist = MvNormal([0,1.], [0.8 0.5; 0.5 1.0])

D = rand(generative_dist, N)                                    # Generate observations from generat
scatter(D[1,:], D[2,:], marker=:x, markerstrokewidth=3, label=L"D")
x_dot = rand(generative_dist)                                   # Generate x·
scatter!([x_dot[1]], [x_dot[2]], label=L"x_\bullet")
plot!(range(0, 2), [1., 1., 1.], fillrange=2, alpha=0.4, color=:gray,label=L"S")
```

Out[2]:



The Gaussian Distribution

- Consider a random (vector) variable $x \in \mathbb{R}^M$ that is "normally" (i.e., Gaussian) distributed. The *moment* parameterization of the Gaussian distribution is completely specified by its *mean* μ and *variance* Σ and given by

$$p(x|\mu,\Sigma) = \mathcal{N}(x|\mu,\Sigma) \triangleq \frac{1}{\sqrt{(2\pi)^M|\Sigma|}} \exp \left\{ -\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu) \right\}.$$

  where $|\Sigma| \triangleq \det(\Sigma)$ is the determinant of Σ.
  - For the scalar real variable $x \in \mathbb{R}$, this works out to

$$p(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left\{ -\frac{(x-\mu)^2}{2\sigma^2} \right\}.$$

- Alternatively, the _canonical_ (a.k.a. _natural_ or _information_) parameterization of the Gaussian distribution is given by

$$p(x|\eta,\Lambda) = \mathcal{N}_c(x|\eta,\Lambda) = \exp \left\{ a + \eta^T x - \frac{1}{2}x^T\Lambda x \right\}.$$

  - $a = -\frac{1}{2}\left(M\log(2\pi) - \log|\Lambda| + \eta^T\Lambda\eta\right)$ is the normalizing constant that ensures that $\int p(x)\mathrm{d}x = 1$.
  - $\Lambda = \Sigma^{-1}$ is called the *precision matrix*.
  - $\eta = \Sigma^{-1}\mu$ is the *natural* mean or for clarity often called the *precision-weighted* mean.

Why the Gaussian?

- Why is the Gaussian distribution so ubiquitously used in science and engineering? (see also Jaynes, section 7.14, and the whole chapter 7 in his book).

- (1) Operations on probability distributions tend to lead to Gaussian distributions:
  - Any smooth function with single rounded maximum, if raised to higher and higher powers, goes into a Gaussian function. (useful in sequential Bayesian inference).
  - The Gaussian distribution has higher entropy than any other with the same variance.
    - Therefore any operation on a probability distribution that discards information but preserves variance gets us closer to a Gaussian.
    - As an example, see Jaynes, section 7.1.4 for how this leads to the Central Limit Theorem, which results from performing

convolution operations on distributions.

- (2) Once the Gaussian has been attained, this form tends to be preserved. e.g.,
  - The convolution of two Gaussian functions is another Gaussian function (useful in sum of 2 variables and linear transformations)
  - The product of two Gaussian functions is another Gaussian function (useful in Bayes rule).
  - The Fourier transform of a Gaussian function is another Gaussian function.

Transformations and Sums of Gaussian Variables

- A **linear transformation** $z = Ax + b$ of a Gaussian variable $x \sim \mathcal{N}(\mu_x, \Sigma_x)$ is Gaussian distributed as

$$p(z) = \mathcal{N}\left(z \,|\, A\mu_x + b, A\Sigma_x A^T\right) \tag{SRG-4a}$$

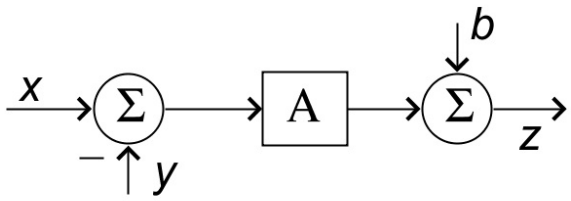
  - In fact, after a linear transformation $z = Ax + b$, no matter how x is distributed, the mean and variance of z are always given by $\mu_z = A\mu_x + b$ and $\Sigma_z = A\Sigma_x A^T$, respectively (see probability theory review lesson). In case x is not Gaussian, higher order moments may be needed to specify the distribution for z.

- The **sum of two independent Gaussian variables** is also Gaussian distributed. Specifically, if $x \sim \mathcal{N}\left(\mu_x, \Sigma_x\right)$ and $y \sim \mathcal{N}\left(\mu_y, \Sigma_y\right)$, then the PDF for $z = x + y$ is given by

$$\begin{aligned} p(z) &= \mathcal{N}(x \,|\, \mu_x, \Sigma_x) * \mathcal{N}(y \,|\, \mu_y, \Sigma_y) \\ &= \mathcal{N}\left(z \,|\, \mu_x + \mu_y, \Sigma_x + \Sigma_y\right) \end{aligned} \tag{SRG-8}$$

  - The sum of two Gaussian *distributions* is NOT a Gaussian distribution. Why not?

Example: Gaussian Signals in a Linear System



- Given independent variables $x \sim \mathcal{N}(\mu_x, \sigma_x^2)$ and $y \sim \mathcal{N}(\mu_y, \sigma_y^2)$, what is the PDF for $z = A \cdot (x - y) + b$? (for answer, see Exercises)

- Think about the role of the Gaussian distribution for stochastic linear systems in relation to what sinusoidals mean for deterministic linear system analysis.

Bayesian Inference for the Gaussian

- Let's estimate a constant θ from one 'noisy' measurement x about that constant.
- We assume the following measurement equations (the tilde $\sim$ means: 'is distributed as'):

$$\begin{aligned} x &= \theta + \epsilon \\ \epsilon &\sim \mathcal{N}(0, \sigma^2) \end{aligned}$$

- Also, let's assume a Gaussian prior for θ

$$\theta \sim \mathcal{N}(\mu_0, \sigma_0^2)$$

Model specification

- Note that you can rewrite these specifications in probabilistic notation as follows:

$$\begin{aligned} p(x|\theta) &= \mathcal{N}(x|\theta, \sigma^2) \\ p(\theta) &= \mathcal{N}(\theta|\mu_0, \sigma_0^2) \end{aligned}$$

- (**Notational convention**). Note that we write $\epsilon \sim \mathcal{N}(0, \sigma^2)$ but not $\epsilon \sim \mathcal{N}(\epsilon|0, \sigma^2)$, and we write $p(\theta) = \mathcal{N}(\theta|\mu_0, \sigma_0^2)$ but not $p(\theta) = \mathcal{N}(\mu_0, \sigma_0^2)$.

Inference

- For simplicity, we assume that the variance σ^2 is given and will proceed to derive a Bayesian posterior for the mean θ. The case for Bayesian inference of σ^2 with a given mean is discussed in the optional slides.

- Let's do Bayes rule for the posterior PDF $p(\theta|x)$.

$$
\begin{aligned}
p(\theta|x) &= \frac{p(x|\theta)p(\theta)}{p(x)} \propto p(x|\theta)p(\theta) \\
&= \mathcal{N}(x|\theta, \sigma^2)\mathcal{N}(\theta|\mu_0, \sigma_0^2) \\
&\propto \exp\left\{-\frac{(x-\theta)^2}{2\sigma^2} - \frac{(\theta-\mu_0)^2}{2\sigma_0^2}\right\} \\
&\propto \exp \\
&\qquad \left\{\theta^2 \cdot \left(-\frac{1}{2\sigma_0^2} - \frac{1}{2\sigma^2}\right) + \theta \cdot \left(\frac{\mu_0}{\sigma_0^2} + \frac{x}{\sigma^2}\right)\right\} \\
&= \exp\left\{-\frac{\sigma_0^2 + \sigma^2}{2\sigma_0^2\sigma^2}\left(\theta - \frac{\sigma_0^2 x + \sigma^2\mu_0}{\sigma^2 + \sigma_0^2}\right)^2\right\}
\end{aligned}
$$

  which we recognize as a Gaussian distribution w.r.t. θ.

- (Just as an aside,) this computational 'trick' for multiplying two Gaussians is called **completing the square**. The procedure makes use of the equality

$$
ax^2 + bx + c_1 = a\left(x + \frac{b}{2a}\right)^2 + c_2
$$

- In particular, it follows that the posterior for θ is

$$
p(\theta|x) = \mathcal{N}(\theta|\,\mu_1, \sigma_1^2)
$$

  where

$$
\begin{aligned}
\frac{1}{\sigma_1^2} &= \frac{\sigma_0^2 + \sigma^2}{\sigma^2\sigma_0^2} = \frac{1}{\sigma_0^2} + \frac{1}{\sigma^2} \\
\mu_1 &= \frac{\sigma_0^2 x + \sigma^2\mu_0}{\sigma^2 + \sigma_0^2} = \sigma_1^2\left(\frac{1}{\sigma_0^2}\mu_0 + \frac{1}{\sigma^2}x\right)
\end{aligned}
$$

(Multivariate) Gaussian Multiplication

- So, multiplication of two Gaussian distributions yields another (unnormalized) Gaussian with
  - posterior precision equals **sum of prior precisions**
  - posterior precision-weighted mean equals **sum of prior precision-weighted means**

- As we just saw, a Gaussian prior, combined with a Gaussian likelihood, make Bayesian inference analytically solvable (!):

$$
\underbrace{\text{Gaussian}}_{\text{posterior}} \propto \underbrace{\text{Gaussian}}_{\text{likelihood}} \times \underbrace{\text{Gaussian}}_{\text{prior}}
$$

- In general, the multiplication of two multi-variate Gaussians over x yields an (unnormalized) Gaussian over x:

$$\boxed{\begin{aligned}&\mathcal{N}(x|\mu_a,\Sigma_a)\cdot\mathcal{N}(x|\mu_b,\Sigma_b)\\&=\underbrace{\mathcal{N}(\mu_a|\,\mu_b,\Sigma_a+\Sigma_b)}_{\text{normalization constant}}\cdot\mathcal{N}(x|\mu_c,\Sigma_c)\end{aligned}} \tag{SRG-6}$$

where

$$\begin{aligned}\Sigma_c^{-1}&=\Sigma_a^{-1}+\Sigma_b^{-1}\\\Sigma_c^{-1}\mu_c&=\Sigma_a^{-1}\mu_a+\Sigma_b^{-1}\mu_b\end{aligned}$$

- Check out that normalization constant $\mathcal{N}(\mu_a|\,\mu_b,\Sigma_a+\Sigma_b)$. Amazingly, this constant can also be expressed by a Gaussian!
- $\Rightarrow$ Note that Bayesian inference is trivial in the *canonical* parameterization of the Gaussian, where we would get

$$\begin{aligned}\Lambda_c&=\Lambda_a+\Lambda_b &&\text{(precisions add)}\\\eta_c&=\eta_a+\eta_b &&\text{(precision-weighted means add)}\end{aligned}$$

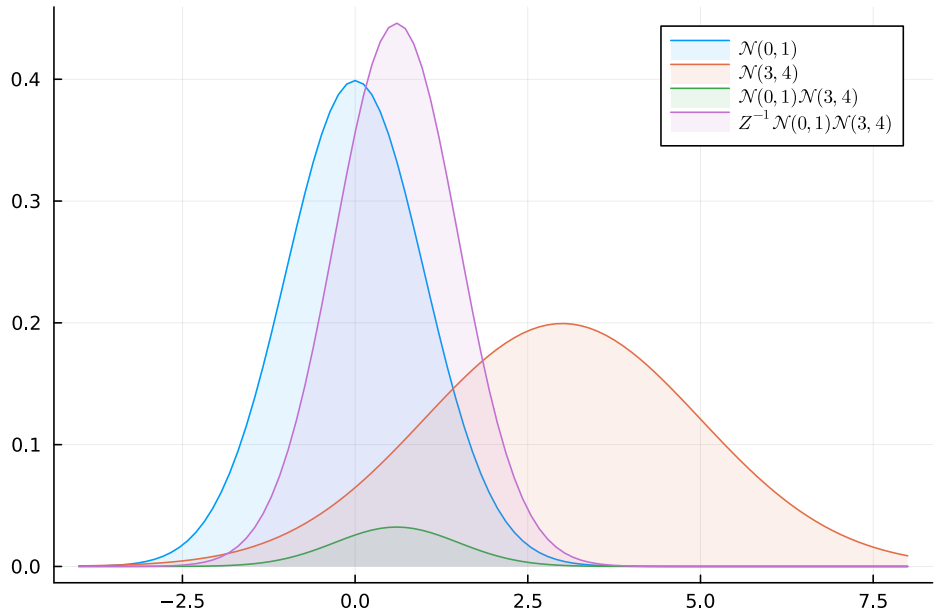
  - This property is an important reason why the canonical parameterization of the Gaussian distribution is useful in Bayesian data processing.

Code Example: Product of Two Gaussian PDFs

- Let's plot the exact product of two Gaussian PDFs as well as the normalized product according to the above derivation.

In [3]:
```
using Plots, Distributions, LaTeXStrings
d1 = Normal(0, 1) # μ=0, σ^2=1
d2 = Normal(3, 2) # μ=3, σ^2=4

# Calculate the parameters of the product d1*d2
s2_prod = (d1.σ^-2 + d2.σ^-2)^-1
m_prod = s2_prod * ((d1.σ^-2)*d1.μ + (d2.σ^-2)*d2.μ)
d_prod = Normal(m_prod, sqrt(s2_prod)) # Note that we neglect the normalization constant.

# Plot stuff
x = range(-4, stop=8, length=100)
plot(x, pdf.(d1,x), label=L"\mathcal{N}(0,1)", fill=(0, 0.1))                                  # Plot the f
plot!(x, pdf.(d2,x), label=L"\mathcal{N}(3,4)", fill=(0, 0.1))                                 # Plot the s
plot!(x, pdf.(d1,x) .* pdf.(d2,x), label=L"\mathcal{N}(0,1) \mathcal{N}(3,4)", fill=(0, 0.1))  # Plot the e
plot!(x, pdf.(d_prod,x), label=L"Z^{-1} \mathcal{N}(0,1) \mathcal{N}(3,4)", fill=(0, 0.1))     # Plot the n
```

Out[3]:



Bayesian Inference with multiple Observations

- Now consider that we measure a data set $D=\{x_1,x_2,\ldots,x_N\}$, with measurements

$$x_n = \theta + \epsilon_n$$
$$\epsilon_n \sim \mathcal{N}(0, \sigma^2)$$

and the same prior for θ:

$$\theta \sim \mathcal{N}(\mu_0, \sigma_0^2)$$

- Let's derive the distribution $p(x_{N+1}|D)$ for the next sample .

inference

- First, we derive the posterior for θ:

$$p(\theta|D) \propto \underbrace{\mathcal{N}(\theta|\mu_0, \sigma_0^2)}_{\text{prior}} \cdot \underbrace{\prod_{n=1}^{N} \mathcal{N}(x_n|\theta, \sigma^2)}_{\text{likelihood}}$$

  which is a multiplication of $N+1$ Gaussians and is therefore also Gaussian-distributed.

- Using the property that precisions and precision-weighted means add when Gaussians are multiplied, we can immediately write the posterior

$$p(\theta|D) = \mathcal{N}(\theta|\, \mu_N, \sigma_N^2)$$

  as

$$\frac{1}{\sigma_N^2} = \frac{1}{\sigma_0^2} + \sum_n \frac{1}{\sigma^2} \qquad \text{(B-2.142)}$$
$$\mu_N = \sigma_N^2 \left(\frac{1}{\sigma_0^2}\mu_0 + \sum_n \frac{1}{\sigma^2} x_n \right) \qquad \text{(B-2.141)}$$

application: prediction of future sample

- We now have a posterior for the model parameters. Let's write down what we know about the next sample x_{N+1}.

$$\begin{aligned}
p(x_{N+1}|D) &= \int p(x_{N+1}|\theta) p(\theta|D) \mathrm{d}\theta \\
&= \int \mathcal{N}(x_{N+1}|\theta, \sigma^2) \mathcal{N}(\theta|\mu_N, \sigma_N^2) \mathrm{d}\theta \\
&= \int \mathcal{N}(\theta|x_{N+1}, \sigma^2) \mathcal{N}(\theta|\mu_N, \sigma_N^2) \mathrm{d}\theta \\
&= \int \mathcal{N}(x_{N+1}|\mu_N, \sigma_N^2 + \sigma^2) \mathcal{N}(\theta|\cdot, \cdot) \mathrm{d}\theta \qquad \text{(use SRG-6)} \\
&= \mathcal{N}(x_{N+1}|\mu_N, \sigma_N^2 + \sigma^2) \underbrace{\int \mathcal{N}(\theta|\cdot, \cdot) \mathrm{d}\theta}_{=1} \\
&= \mathcal{N}(x_{N+1}|\mu_N, \sigma_N^2 + \sigma^2)
\end{aligned}$$

- Uncertainty about x_{N+1} involved both uncertainty about the parameter (σ_N^2) and observation noise σ^2.

Maximum Likelihood Estimation for the Gaussian

- In order to determine the *maximum likelihood* estimate of θ, we let $\sigma_0^2 \to \infty$ (leads to uniform prior for θ), yielding $\frac{1}{\sigma_N^2} = \frac{N}{\sigma^2}$ and consequently

$$\mu_{\mathrm{ML}} = \mu_N|_{\sigma_0^2 \to \infty} = \sigma_N^2 \left(\frac{1}{\sigma^2} \sum_n x_n \right) = \frac{1}{N} \sum_{n=1}^{N} x_n$$

- As expected, having an expression for the maximum likelihood estimate, it is now possible to rewrite the (Bayesian) posterior mean for θ as

$$\begin{aligned} \underbrace{\mu_N}_{\text{posterior}} &= \sigma_N^2 \left(\frac{1}{\sigma_0^2} \mu_0 + \sum_n \frac{1}{\sigma^2} x_n \right) \\ &= \frac{\sigma_0^2 \sigma^2}{N\sigma_0^2 + \sigma^2} \left(\frac{1}{\sigma_0^2} \mu_0 + \sum_n \frac{1}{\sigma^2} x_n \right) \\ &= \frac{\sigma^2}{N\sigma_0^2 + \sigma^2} \mu_0 + \frac{N\sigma_0^2}{N\sigma_0^2 + \sigma^2} \mu_{\mathrm{ML}} \\ &= \underbrace{\mu_0}_{\text{prior}} + \underbrace{\underbrace{\frac{N\sigma_0^2}{N\sigma_0^2 + \sigma^2}}_{\text{gain}} \cdot \underbrace{(\mu_{\mathrm{ML}} - \mu_0)}_{\text{prediction error}}}_{\text{correction}} \end{aligned} \tag{B-2.141}$$

- Hence, the posterior mean always lies somewhere between the prior mean μ_0 and the maximum likelihood estimate (the "data" mean) μ_{ML}.

Conditioning and Marginalization of a Gaussian

- Let $z = \begin{bmatrix} x \\ y \end{bmatrix}$ be jointly normal distributed as

$$\begin{aligned} p(z) &= \mathcal{N}(z|\mu, \Sigma) \\ &= \mathcal{N}\left(\begin{bmatrix} x \\ y \end{bmatrix} \middle| \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}, \begin{bmatrix} \Sigma_x & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_y \end{bmatrix} \right) \end{aligned}$$

- Since covariance matrices are by definition symmetric, it follows that Σ_x and Σ_y are symmetric and $\Sigma_{xy} = \Sigma_{yx}^T$.
- Let's factorize $p(z) = p(x, y)$ as $p(x, y) = p(y|x)p(x)$ through conditioning and marginalization.

conditioning: $\boxed{\begin{aligned} &p(y|x) \\ &= \mathcal{N}\left(y \,|\, \mu_y + \Sigma_{yx}\Sigma_x^{-1}(x - \mu_x),\, \Sigma_y - \Sigma_{yx}\Sigma_x^{-1}\Sigma_{xy}\right) \end{aligned}}$

$$\text{marginalization: } \boxed{p(x) = \mathcal{N}\left(x|\mu_x, \Sigma_x\right)}$$

- **proof**: in Bishop pp.87-89
- Hence, conditioning and marginalization in Gaussians leads to Gaussians again. This is very useful for applications to Bayesian inference in jointly Gaussian systems.
- With a natural parameterization of the Gaussian $p(z) = \mathcal{N}_c(z|\eta, \Lambda)$ with precision matrix $\Lambda = \Sigma^{-1} = \begin{bmatrix} \Lambda_x & \Lambda_{xy} \\ \Lambda_{yx} & \Lambda_y \end{bmatrix}$, the conditioning operation results in a simpler result, see Bishop pg.90, eqs. 2.96 and 2.97.

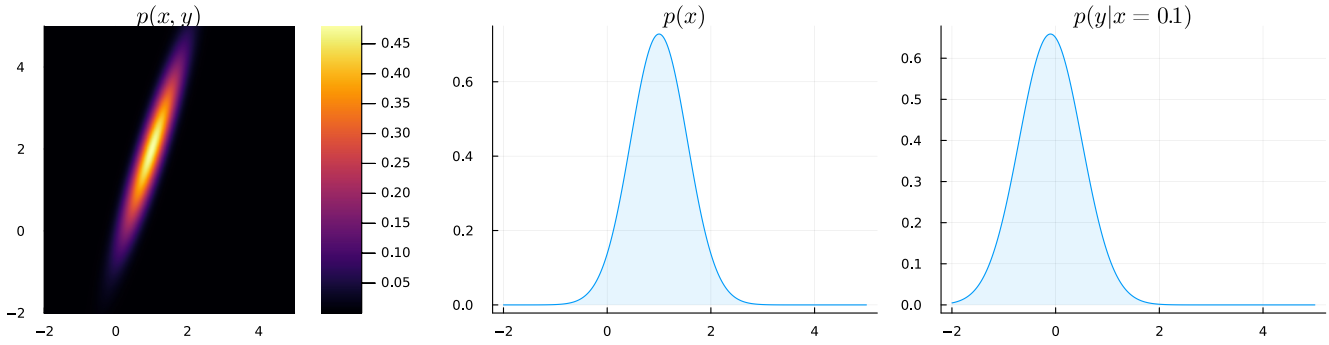
- As an exercise, interpret the formula for the conditional mean ($\mathbb{E}[y|x] = \mu_y + \Sigma_{yx}\Sigma_x^{-1}(x - \mu_x)$) as a prediction-correction operation.

Code Example: Joint, Marginal, and Conditional Gaussian Distributions

- Let's plot of the joint, marginal, and conditional distributions.

In [4]:
```
using Plots, LaTeXStrings, Distributions

# Define the joint distribution p(x,y)
μ = [1.0; 2.0]
Σ = [0.3 0.7;
     0.7 2.0]
joint = MvNormal(μ,Σ)

# Define the marginal distribution p(x)
marginal_x = Normal(μ[1], sqrt(Σ[1,1]))

# Plot p(x,y)
x_range = y_range = range(-2,stop=5,length=1000)
joint_pdf = [ pdf(joint, [x_range[i];y_range[j]]) for  j=1:length(y_range), i=1:length(x_range)]
plot_1 = heatmap(x_range, y_range, joint_pdf, title = L"p(x, y)")

# Plot p(x)
plot_2 = plot(range(-2,stop=5,length=1000), pdf.(marginal_x, range(-2,stop=5,length=1000)), title = L"p(x)",

# Plot p(y|x = 0.1)
x = 0.1
conditional_y_m = μ[2]+Σ[2,1]*inv(Σ[1,1])*(x-μ[1])
conditional_y_s2 = Σ[2,2] - Σ[2,1]*inv(Σ[1,1])*Σ[1,2]
conditional_y = Normal(conditional_y_m, sqrt.(conditional_y_s2))
plot_3 = plot(range(-2,stop=5,length=1000), pdf.(conditional_y, range(-2,stop=5,length=1000)), title = L"p(y
plot(plot_1, plot_2, plot_3, layout=(1,3), size=(1200,300))
```

Out[4]:



As is clear from the plots, the conditional distribution is a renormalized slice from the joint distribution.

Example: Conditioning of Gaussian

- Consider (again) the system

$$p(x \mid \theta) = \mathcal{N}(x \mid \theta, \sigma^2)$$
$$p(\theta) = \mathcal{N}(\theta \mid \mu_0, \sigma_0^2)$$

- Let $z = \begin{bmatrix} x \\ \theta \end{bmatrix}$. The distribution for z is then given by (Exercise)

$$p(z) = p\left(\begin{bmatrix} x \\ \theta \end{bmatrix}\right)$$
$$= \mathcal{N}\left(\begin{bmatrix} x \\ \theta \end{bmatrix} \middle| \begin{bmatrix} \mu_0 \\ \mu_0 \end{bmatrix}, \begin{bmatrix} \sigma_0^2 + \sigma^2 & \sigma_0^2 \\ \sigma_0^2 & \sigma_0^2 \end{bmatrix}\right)$$

- Direct substitution of the rule for Gaussian conditioning leads to the posterior (derivation as an Exercise):

$$p(\theta|x) = \mathcal{N}\left(\theta \,|\, \mu_1, \sigma_1^2\right) \,,$$

with

$$\begin{aligned} K &= \frac{\sigma_0^2}{\sigma_0^2 + \sigma^2} \qquad (K \text{ is called: Kalman gain}) \\ \mu_1 &= \mu_0 + K \cdot (x - \mu_0) \\ \sigma_1^2 &= (1 - K)\, \sigma_0^2 \end{aligned}$$

- $\Rightarrow$ Moral: For jointly Gaussian systems, we can do inference simply in one step by using the formulas for conditioning and marginalization.

Recursive Bayesian Estimation for Adaptive Signal Processing

- Consider the signal $x_t = \theta + \epsilon_t$, where $D_t = \{x_1, \ldots, x_t\}$ is observed *sequentially* (over time).
- **Problem**: Derive a recursive algorithm for $p(\theta|D_t)$, i.e., an update rule for (posterior) $p(\theta|D_t)$ based on (prior) $p(\theta|D_{t-1})$ and (new observation) x_t.

Model specification

- Let's define the estimate after t observations (i.e., our *solution*) as $p(\theta|D_t) = \mathcal{N}(\theta \,|\, \mu_t, \sigma_t^2)$.
- We define the joint distribution for θ and x_t, given background D_{t-1}, by

$$\begin{aligned} p(x_t, \theta \,|\, D_{t-1}) &= p(x_t|\theta)\, p(\theta|D_{t-1}) \\ &= \underbrace{\mathcal{N}(x_t \,|\, \theta, \sigma^2)}_{\text{likelihood}} \underbrace{\mathcal{N}(\theta \,|\, \mu_{t-1}, \sigma_{t-1}^2)}_{\text{prior}} \end{aligned}$$

Inference

- Use Bayes rule,

$$\begin{aligned} p(\theta|D_t) &= p(\theta|x_t, D_{t-1}) \\ &\propto p(x_t, \theta|D_{t-1}) \\ &= p(x_t|\theta)\, p(\theta|D_{t-1}) \\ &= \mathcal{N}(x_t|\theta, \sigma^2)\, \mathcal{N}(\theta \,|\, \mu_{t-1}, \sigma_{t-1}^2) \\ &= \mathcal{N}(\theta|x_t, \sigma^2)\, \mathcal{N}(\theta \,|\, \mu_{t-1}, \sigma_{t-1}^2) \\ &\qquad (\text{note this trick}) \\ &= \mathcal{N}(\theta|\mu_t, \sigma_t^2) \\ &\qquad (\text{use Gaussian multiplication formula SRG-6}) \end{aligned}$$

with

$$\begin{aligned} K_t &= \frac{\sigma_{t-1}^2}{\sigma_{t-1}^2 + \sigma^2} \qquad (\text{Kalman gain}) \\ \mu_t &= \mu_{t-1} + K_t \cdot (x_t - \mu_{t-1}) \\ \sigma_t^2 &= (1 - K_t)\, \sigma_{t-1}^2 \end{aligned}$$

- This linear *sequential* estimator of mean and variance in Gaussian observations is called a **Kalman Filter**.
- The so-called Kalman gain K_t serves as a "learning rate" (step size) in the parameter update equation $\mu_t = \mu_{t-1} + K_t \cdot (x_t - \mu_{t-1})$. Note that *you* don't need to choose the learning rate. Bayesian inference computes its own (optimal) learning rates.

- Note that the uncertainty about θ decreases over time (since $0 < (1 - K_t) < 1$). If we assume that the statistics of the system do not change (stationarity), each new sample provides new information about the process, so the uncertainty decreases.

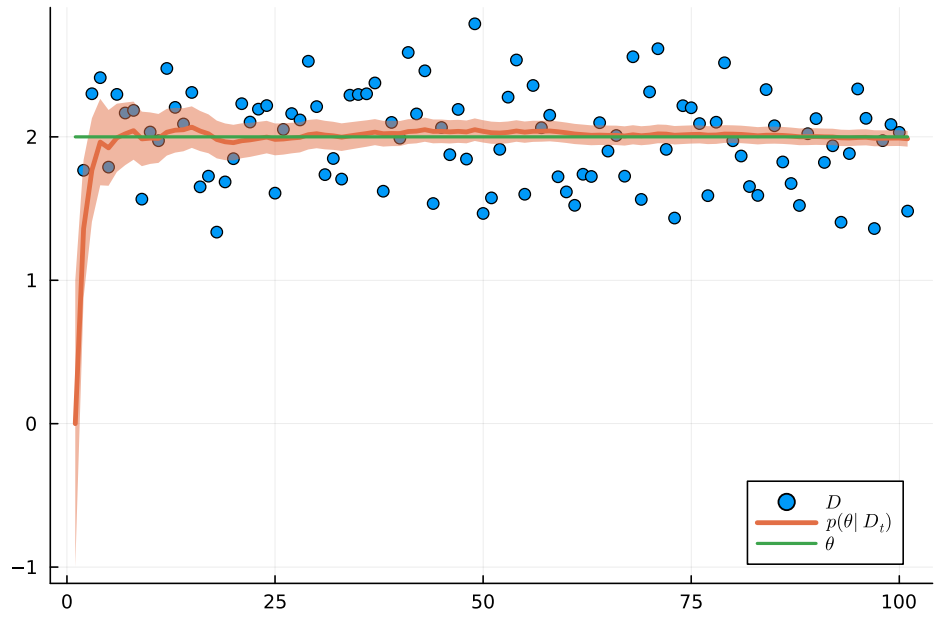
- Recursive Bayesian estimation as discussed here is the basis for **adaptive signal processing** algorithms such as Least Mean Squares (LMS) and Recursive Least Squares (RLS). Both RLS and LMS are special cases of Recursive Bayesian estimation.

Code Example: Kalman Filter

- Let's implement the Kalman filter described above. We'll use it to recursively estimate the value of θ based on noisy observations.

In [5]:
```
using Plots, Distributions

n = 100          # specify number of observations
θ = 2.0          # true value of the parameter we would like to estimate
noise_σ2 = 0.3   # variance of observation noise

observations = noise_σ2 * randn(n) .+ θ

function perform_kalman_step(prior :: Normal, x :: Float64, noise_σ2 :: Float64)
    K = prior.σ / (noise_σ2 + prior.σ)           # compute the Kalman gain
    posterior_μ = prior.μ + K*(x - prior.μ)      # update the posterior mean
    posterior_σ = prior.σ * (1.0 - K)            # update the posterior standard deviation
    return Normal(posterior_μ, posterior_σ)      # return the posterior distribution
end

post_μ = fill!(Vector{Float64}(undef,n + 1), NaN)      # means of p(θ|D) over time
post_σ2 = fill!(Vector{Float64}(undef,n + 1), NaN)     # variances of p(θ|D) over time

prior = Normal(0, 1)     # specify the prior distribution (you can play with the parameterization of this to

post_μ[1] = prior.μ      # save prior mean and variance to show these in plot
post_σ2[1] = prior.σ

for (i, x) in enumerate(observations)                          # note that this loop demonstrates Bayesian
    posterior = perform_kalman_step(prior, x, noise_σ2)        # compute the posterior distribution given t
    post_μ[i + 1] = posterior.μ                                # save the mean of the posterior distributic
    post_σ2[i + 1] = posterior.σ                               # save the variance of the posterior distrib
    prior = posterior                                          # the posterior becomes the prior for future
end

obs_scale = collect(2:n+1)
scatter(obs_scale, observations, label=L"D", )
post_scale = collect(1:n+1)                                                                  # scatter the observatic
plot!(post_scale, post_μ, ribbon=sqrt.(post_σ2), linewidth=3, label=L"p(θ | D_t)")           # lineplot our estimatec
plot!(post_scale, θ*ones(n + 1), linewidth=2, label=L"θ")                                    # plot the true value of
```

Out[5]:



The shaded area represents 2 standard deviations of posterior $p(\theta|D)$. The variance of the posterior is guaranteed to decrease monotonically for the standard Kalman filter.

Product of Normally Distributed Variables

- (We've seen that) the sum of two Gausssian distributed variables is also Gaussian distributed.
- Has the *product* of two Gaussian distributed variables also a Gaussian distribution?
- **No**! In general this is a difficult computation. As an example, let's compute $p(z)$ for $Z = XY$ for the special case that $X \sim \mathcal{N}(0,1)$ and $Y \sim \mathcal{N}(0,1)$.

$$\begin{aligned} p(z) &= \int_{X,Y} p(z|x,y)\, p(x,y)\, \mathrm{d}x\mathrm{d}y \\ &= \frac{1}{2\pi} \int \delta(z - xy)\, e^{-(x^2+y^2)/2}\, \mathrm{d}x\mathrm{d}y \\ &= \frac{1}{\pi} \int_0^\infty \frac{1}{x} e^{-(x^2+z^2/x^2)/2}\, \mathrm{d}x \\ &= \frac{1}{\pi} \mathrm{K}_0(|z|)\,. \end{aligned}$$

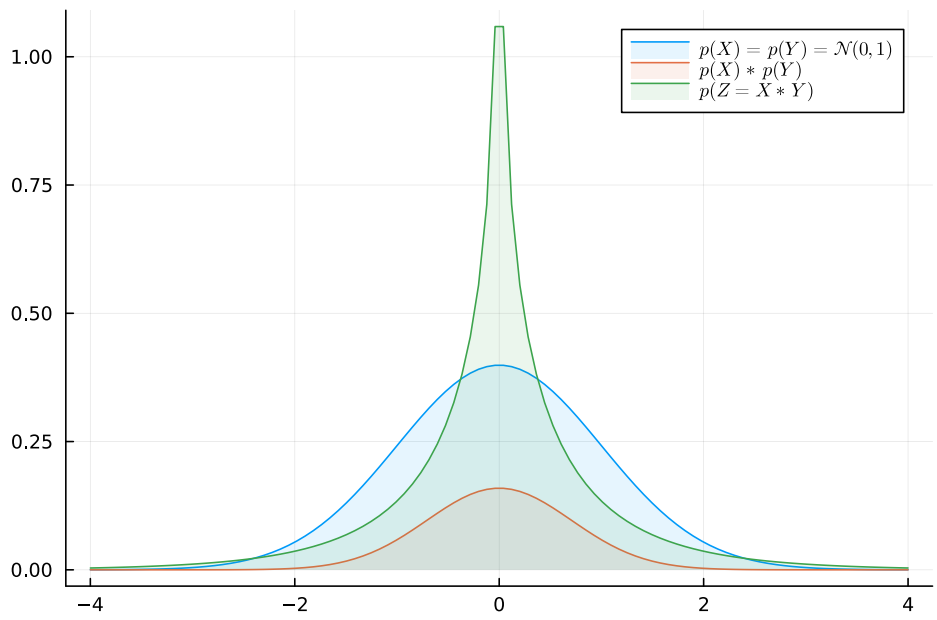
where $\mathrm{K}_n(z)$ is a modified Bessel function of the second kind.

Code Example: Product of Gaussian Distributions

- We plot $p(Z = XY)$ and $p(X)p(Y)$ for $X \sim \mathcal{N}(0,1)$ and $Y \sim \mathcal{N}(0,1)$ to give an idea of how these distributions differ.

In [6]:
```
using Plots, Distributions, SpecialFunctions, LaTeXStrings
X = Normal(0,1)
Y = Normal(0,1)
pdf_product_std_normals(z::Vector) = (besselk.(0, abs.(z))./π)
range1 = collect(range(-4,stop=4,length=100))
plot(range1, pdf.(X, range1), label=L"p(X)=p(Y)=\mathcal{N}(0,1)", fill=(0, 0.1))
plot!(range1, pdf.(X,range1).*pdf.(Y,range1), label=L"p(X)*p(Y)", fill=(0, 0.1))
plot!(range1, pdf_product_std_normals(range1), label=L"p(Z=X*Y)", fill=(0, 0.1))
```

Out[6]:



- In short, Gaussian-distributed variables remain Gaussian in linear systems, but this is not the case in non-linear systems.

Solution to Example Problem

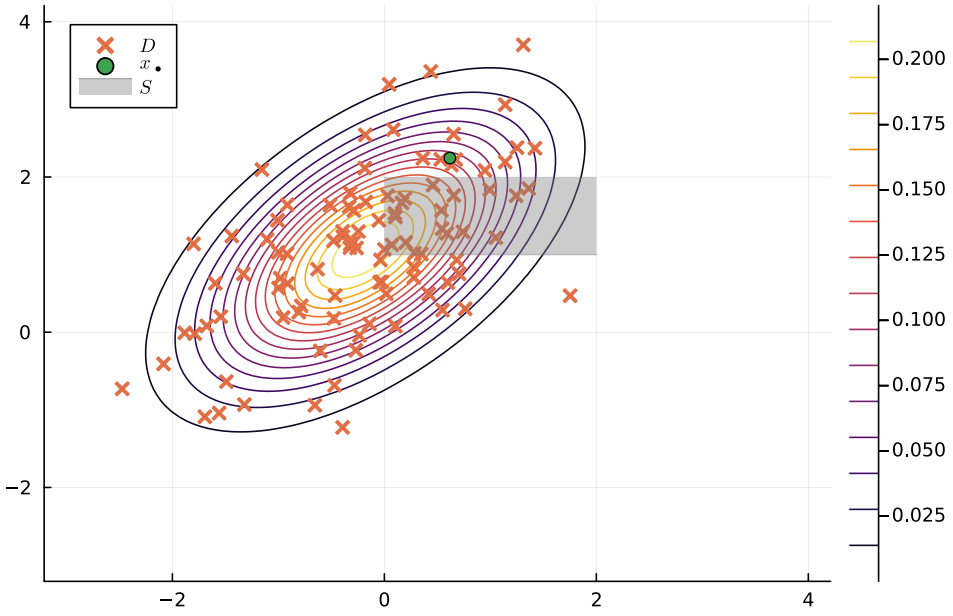
We apply maximum likelihood estimation to fit a 2-dimensional Gaussian model (m) to data set D. Next, we evaluate $p(x_\bullet \in S|m)$ by (numerical) integration of the Gaussian pdf over S: $p(x_\bullet \in S|m) = \int_S p(x|m)\mathrm{d}x$.

In [7]:
```
using HCubature, LinearAlgebra, Plots, Distributions# Numerical integration package
```

```
# Maximum likelihood estimation of 2D Gaussian
N = length(sum(D,dims=1))
μ = 1/N * sum(D,dims=2)[:,1]
D_min_μ = D - repeat(μ, 1, N)
Σ = Hermitian(1/N * D_min_μ*D_min_μ')
m = MvNormal(μ, convert(Matrix, Σ));

contour(range(-3, 4, length=100), range(-3, 4, length=100), (x, y) -> pdf(m, [x, y]))

# Numerical integration of p(x|m) over S:
(val,err) = hcubature((x)->pdf(m,x), [0., 1.], [2., 2.])
println("p(x∈S|m) ≈ $(val)")

scatter!(D[1,:], D[2,:], marker=:x, markerstrokewidth=3, label=L"D")
scatter!([x_dot[1]], [x_dot[2]], label=L"x_\bullet")
plot!(range(0, 2), [1., 1., 1.], fillrange=2, alpha=0.4, color=:gray, label=L"S")
```

```
p(x∈S|m) ≈ 0.17884370364565447
```

Out[7]:



Summary

1. A **linear transformation** $z = Ax + b$ of a Gaussian variable $x \sim \mathcal{N}(\mu_x, \Sigma_x)$ is Gaussian distributed as

$$p(z) = \mathcal{N}\left(z \,|\, A\mu_x + b, A\Sigma_x A^T\right)$$

2. Bayesian inference with a Gaussian prior and Gaussian likelihood leads to an analytically computable Gaussian posterior, because of the **multiplication rule for Gaussians**:

$$\begin{aligned} &\mathcal{N}(x|\mu_a, \Sigma_a) \cdot \mathcal{N}(x|\mu_b, \Sigma_b) \\ &= \underbrace{\mathcal{N}(\mu_a|\, \mu_b, \Sigma_a + \Sigma_b)}_{\text{normalization constant}} \cdot \mathcal{N}(x|\mu_c, \Sigma_c) \end{aligned}$$

where

$$\begin{aligned} \Sigma_c^{-1} &= \Sigma_a^{-1} + \Sigma_b^{-1} \\ \Sigma_c^{-1}\mu_c &= \Sigma_a^{-1}\mu_a + \Sigma_b^{-1}\mu_b \end{aligned}$$

3. **Conditioning and marginalization** of a multivariate Gaussian distribution yields Gaussian distributions. In particular, the joint distribution

$$\mathcal{N}\left(\begin{bmatrix} x \\ y \end{bmatrix} \middle| \begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}, \begin{bmatrix} \Sigma_x & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_y \end{bmatrix}\right)$$

can be decomposed as

$$p(y|x) = \mathcal{N}\left(y \,|\, \mu_y + \Sigma_{yx}\Sigma_x^{-1}(x-\mu_x),\ \Sigma_y - \Sigma_{yx}\Sigma_x^{-1}\Sigma_{xy}\right)$$
$$p(x) = \mathcal{N}\left(x|\mu_x, \Sigma_x\right)$$

- Here's a nice summary of Gaussian calculations by Sam Roweis.

OPTIONAL SLIDES

Inference for the Precision Parameter of the Gaussian

- Again, we consider an observed data set $D = \{x_1, x_2, \ldots, x_N\}$ and try to explain these data by a Gaussian distribution.
- We discussed earlier Bayesian inference for the mean with a given variance. Now we will derive a posterior for the variance if the mean is given. (Technically, we will do the derivation for a precision parameter $\lambda = \sigma^{-2}$, since the discussion is a bit more straightforward for the precision parameter).

model specification

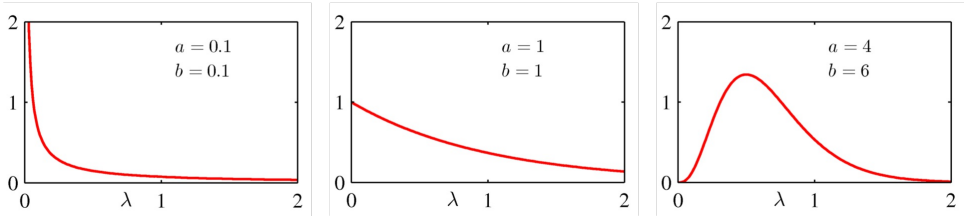
- The likelihood for the precision parameter is

$$p(D|\lambda) = \prod_{n=1}^{N} \mathcal{N}\left(x_n \,|\, \mu, \lambda^{-1}\right)$$
$$\propto \lambda^{N/2} \exp\left\{-\frac{\lambda}{2}\sum_{n=1}^{N}(x_n-\mu)^2\right\} \tag{B-2.145}$$

- The conjugate distribution for this function of λ is the *Gamma* distribution, given by

$$p(\lambda \,|\, a, b) = \mathrm{Gam}\left(\lambda \,|\, a, b\right) \tag{B-2.146}$$
$$\triangleq \frac{1}{\Gamma(a)} b^a \lambda^{a-1} \exp\{-b\lambda\},$$

  where $a > 0$ and $b > 0$ are known as the *shape* and *rate* parameters, respectively.



- (Bishop fig.2.13). Plots of the Gamma distribution $\mathrm{Gam}\left(\lambda \,|\, a, b\right)$ for different values of a and b.
- The mean and variance of the Gamma distribution evaluate to $\mathrm{E}\left(\lambda\right) = \frac{a}{b}$ and $\mathrm{var}\left[\lambda\right] = \frac{a}{b^2}$.

inference

- We will consider a prior $p(\lambda) = \mathrm{Gam}\left(\lambda \,|\, a_0, b_0\right)$, which leads by Bayes rule to the posterior

$$p(\lambda \,|\, D) \propto \underbrace{\lambda^{N/2} \exp\left\{-\frac{\lambda}{2}\sum_{n=1}^{N}(x_n-\mu)^2\right\}}_{\text{likelihood}}$$
$$\cdot \underbrace{\frac{1}{\Gamma(a_0)} b_0^{a_0} \lambda^{a_0-1} \exp\{-b_0\lambda\}}_{\text{prior}}$$
$$\propto \mathrm{Gam}\left(\lambda \,|\, a_N, b_N\right)$$

with

$$a_N = a_0 + \frac{N}{2} \qquad \text{(B-2.150)}$$

$$b_N = b_0 + \frac{1}{2}\sum_n (x_n - \mu)^2 \qquad \text{(B-2.151)}$$

- Hence the **posterior is again a Gamma distribution**. By inspection of B-2.150 and B-2.151, we deduce that we can interpret $2a_0$ as the number of a priori (pseudo-)observations.
- Since the most uninformative prior is given by $a_0 = b_0 \to 0$, we can derive the **maximum likelihood estimate** for the precision as

$$\lambda_{\mathrm{ML}} = \mathrm{E}\left[\lambda\right]|_{a_0=b_0\to 0} = \left.\frac{a_N}{b_N}\right|_{a_0=b_0\to 0} = \frac{N}{\sum_{n=1}^N (x_n - \mu)^2}$$

- In short, if we do density estimation with a Gaussian distribution $\mathcal{N}\left(x_n \,|\, \mu, \sigma^2\right)$ for an observed data set $D = \{x_1, x_2, \ldots, x_N\}$, the maximum likelihood estimates for μ and σ^2 are given by

$$\mu_{\mathrm{ML}} = \frac{1}{N}\sum_{n=1}^N x_n \qquad \text{(B-2.121)}$$

$$\sigma^2_{\mathrm{ML}} = \frac{1}{N}\sum_{n=1}^N (x_n - \mu_{\mathrm{ML}})^2 \qquad \text{(B-2.122)}$$

- These estimates are also known as the *sample mean* and *sample variance* respectively.

Discrete Data and the Multinomial Distribution

Preliminaries

- Goal
  - Simple Bayesian and maximum likelihood-based density estimation for discretely valued data sets
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 67-70, 74-76, 93-94

Discrete Data: the 1-of-K Coding Scheme

- Consider a coin-tossing experiment with outcomes $x \in \{0, 1\}$ (tail and head) and let $0 \leq \mu \leq 1$ represent the probability of heads. This model can written as a **Bernoulli distribution**:

$$p(x|\mu) = \mu^x (1-\mu)^{1-x}$$

  - Note that the variable x acts as a (binary) **selector** for the tail or head probabilities. Think of this as an 'if'-statement in programming.
- Now consider a K-sided coin (e.g., a six-faced *die* (pl.: dice)). How should we encode outcomes?
- **Option 1**: $x \in \{1, 2, \ldots, K\}$.
  - E.g., for $K = 6$, if the die lands on the 3rd face $\Rightarrow x = 3$.
- **Option 2**: $x = (x_1, \ldots, x_K)^T$ with **binary selection variables**

$$x_k = \begin{cases} 1 & \text{if die landed on } k\text{th face} \\ 0 & \text{otherwise} \end{cases}$$

  - E.g., for $K = 6$, if the die lands on the 3rd face $\Rightarrow x = (0, 0, 1, 0, 0, 0)^T$.

- This coding scheme is called a **1-of-K** or **one-hot** coding scheme.

- It turns out that the one-hot coding scheme is mathematically more convenient!

The Categorical Distribution

- Consider a K-sided die. We use a one-hot coding scheme. Assume the probabilities $p(x_k = 1) = \mu_k$ with $\sum_k \mu_k = 1$. The data generating distribution is then (note the similarity to the Bernoulli distribution)

$$p(x|\mu) = \mu_1^{x_1}\mu_2^{x_2}\cdots\mu_K^{x_K} = \prod_{k=1}^{K}\mu_k^{x_k} \tag{B-2.26}$$

- This generalized Bernoulli distribution is called the **categorical distribution** (or sometimes the 'multi-noulli' distribution).

Bayesian Density Estimation for a Loaded Die

- Now let's proceed with Bayesian density estimation, i.e., let's learn the parameters for model $p(x|\theta)$ for an observed data set $D = \{x_1, \ldots, x_N\}$ of N independent-and-identically-distributed (IID) rolls of a K-sided die, with

$$x_{nk} = \begin{cases} 1 & \text{if the } n\text{th throw landed on } k\text{th face} \\ 0 & \text{otherwise} \end{cases}$$

Model specification

- The data generating PDF is

$$p(D|\mu) = \prod_n\prod_k \mu_k^{x_{nk}} = \prod_k \mu_k^{\sum_n x_{nk}} = \prod_k \mu_k^{m_k} \tag{B-2.29}$$

  where $m_k = \sum_n x_{nk}$ is the total number of occurrences that we 'threw' k eyes. Note that $\sum_k m_k = N$.
  - This distribution depends on the observations **only** through the quantities $\{m_k\}$.

- We need a prior for the parameters $\mu = (\mu_1, \mu_2, \ldots, \mu_K)$. In the binary coin toss example, we used a beta distribution that was conjugate with the binomial and forced us to choose prior pseudo-counts.

- The generalization of the beta prior to the K parameters $\{\mu_k\}$ is the Dirichlet distribution:

$$p(\mu|\alpha) = \mathrm{Dir}(\mu|\alpha) = \frac{\Gamma\left(\sum_k \alpha_k\right)}{\Gamma(\alpha_1)\cdots\Gamma(\alpha_K)} \prod_{k=1}^{K}\mu_k^{\alpha_k - 1}$$

  where $\Gamma(\cdot)$ is the Gamma function.

- The Gamma function can be interpreted as a generalization of the factorial function to the real ($\mathbb{R}$) numbers. If n is a natural number ($1, 2, 3, \ldots$), then $\Gamma(n) = (n-1)!$, where $(n-1)! = (n-1)\cdot(n-2)\cdot 1$.

- As before for the Beta distribution in the coin toss experiment, you can interpret $\alpha_k - 1$ as the prior number of (pseudo-)observations that the die landed on the k-th face.

Inference for $\{\mu_k\}$

- The posterior for $\{\mu_k\}$ can be obtained through Bayes rule:

$$
\begin{aligned}
p(\mu|D,\alpha) &\propto p(D|\mu)\cdot p(\mu|\alpha) \\
&\propto \prod_k \mu_k^{m_k} \cdot \prod_k \mu_k^{\alpha_k - 1} \\
&= \prod_k \mu_k^{\alpha_k + m_k - 1} \\
&\propto \operatorname{Dir}(\mu \,|\, \alpha + m) \qquad \text{(B-2.41)} \\
&= \frac{\Gamma\left(\sum_k (\alpha_k + m_k)\right)}{\Gamma(\alpha_1 + m_1)\Gamma(\alpha_2 + m_2)\cdots\Gamma(\alpha_K + m_K)} \\
&\quad \prod_{k=1}^{K} \mu_k^{\alpha_k + m_k - 1}
\end{aligned}
$$

where $m = (m_1, m_2, \ldots, m_K)^T$.

- We recognize the $(\alpha_k - 1)$'s as prior pseudo-counts and the Dirichlet distribution shows to be a conjugate prior to the categorical/multinomial:

$$
\underbrace{\text{Dirichlet}}_{\text{posterior}} \propto \underbrace{\text{categorical}}_{\text{likelihood}} \cdot \underbrace{\text{Dirichlet}}_{\text{prior}}
$$

- This is actually a generalization of the conjugate relation that we found for the binary coin toss:

$$
\underbrace{\text{Beta}}_{\text{posterior}} \propto \underbrace{\text{binomial}}_{\text{likelihood}} \cdot \underbrace{\text{Beta}}_{\text{prior}}
$$

Prediction of next toss for the loaded die

- Let's apply what we have learned about the loaded die to compute the probability that we throw the k-th face at the next toss.

$$
\begin{aligned}
p(x_{\bullet,k} = 1|D) &= \int p(x_{\bullet,k} = 1|\mu)\, p(\mu|D)\, \mathrm{d}\mu \\
&= \int_0^1 \mu_k \times \mathcal{D}ir(\mu|\, \alpha + m)\, \mathrm{d}\mu \\
&= \mathrm{E}\left[\mu_k\right] \\
&= \frac{m_k + \alpha_k}{N + \sum_k \alpha_k}
\end{aligned}
$$

- (You can find the mean of the Dirichlet distribution at its Wikipedia site).
- This result is simply a generalization of **Laplace's rule of succession**.

Categorical, Multinomial and Related Distributions

- In the above derivation, we noticed that the data generating distribution for N die tosses $D = \{x_1, \ldots, x_N\}$ only depends on the **data frequencies** m_k:

$$
p(D|\mu) = \prod_n \underbrace{\prod_k \mu_k^{x_{nk}}}_{\text{categorical dist.}} = \prod_k \mu_k^{\sum_n x_{nk}} = \qquad \text{(B-2.29)}
$$

$$
\prod_k \mu_k^{m_k}
$$

- A related distribution is the distribution over data frequency observations $D_m = \{m_1, \ldots, m_K\}$, which is called the **multinomial distribution**,

$$p(D_m|\mu) = \frac{N!}{m_1!m_2!\ldots m_K!}\prod_k \mu_k^{m_k}.$$

- When used as a likelihood function for μ, it makes no difference whether you use $p(D|\mu)$ or $p(D_m|\mu)$. Why?

- Verify for yourself that (Exercise):
  - the categorial distribution is a special case of the multinomial for $N = 1$.
  - the Bernoulli is a special case of the categorial distribution for $K = 2$.
  - the binomial is a special case of the multinomial for $K = 2$.

Maximum Likelihood Estimation for the Multinomial

Maximum likelihood as a special case of Bayesian estimation

- We can get the maximum likelihood estimate $\hat{\mu}_k$ for μ_k based on N throws of a K-sided die from the Bayesian framework by using a uniform prior for μ and taking the mode of the posterior for μ:

$$\begin{aligned}
\hat{\mu}_k &= \arg\max_{\mu_k} p(D|\mu) \\
&= \arg\max_{\mu_k} p(D|\mu) \cdot \mathrm{Uniform}(\mu) \\
&= \arg\max_{\mu_k} p(D|\mu) \cdot \mathrm{Dir}(\mu|\alpha)|_{\alpha=(1,1,\ldots,1)} \\
&= \arg\max_{\mu_k} p(\mu|D,\alpha)|_{\alpha=(1,1,\ldots,1)} \\
&= \arg\max_{\mu_k} \mathrm{Dir}\left(\mu|m+\alpha\right)|_{\alpha=(1,1,\ldots,1)} \\
&= \frac{m_k}{\sum_k m_k} = \frac{m_k}{N}
\end{aligned}$$

  where we used the fact that the maximum of the Dirichlet distribution $\mathrm{Dir}(\{\alpha_1,\ldots,\alpha_K\})$ is obtained at $(\alpha_k - 1)/(\sum_k \alpha_k - K)$.

Maximum likelihood estimation by optimizing a constrained log-likelihood

- Of course, we shouldn't have to go through the full Bayesian framework to get the maximum likelihood estimate. Alternatively, we can find the maximum likelihood (ML) solution directly by optimizing the (constrained) log-likelihood.

- The log-likelihood for the multinomial distribution is given by

$$\mathrm{L}(\mu) \triangleq \log p(D_m|\mu) \propto \log \prod_k \mu_k^{m_k} = \sum_k m_k \log \mu_k$$

- When doing ML estimation, we must obey the constraint $\sum_k \mu_k = 1$, which can be accomplished by a Lagrange multiplier. The **constrained log-likelihood** with Lagrange multiplier is then

$$\tilde{\mathrm{L}}(\mu) = \sum_k m_k \log \mu_k + \lambda \cdot \left(1 - \sum_k \mu_k\right)$$

- The method of Lagrange multipliers is a mathematical method for transferring a constrained optimization problem to an unconstrained optimization problem (see Bishop App.E). Unconstrained optimization problems can be solved by setting the derivative to zero.

- Setting the derivative of $\tilde{\mathrm{L}}(\mu)$ to zero yields the **sample proportion** for μ_k

$$\nabla_{\mu_k}\tilde{\mathrm{L}}(\mu) = \frac{m_k}{\hat{\mu}_k} - \lambda \stackrel{!}{=} 0 \;\Rightarrow\; \hat{\mu}_k = \frac{m_k}{N}$$

  where we get λ from the constraint

$$\sum_k \hat{\mu}_k = \sum_k \frac{m_k}{\lambda} = \frac{N}{\lambda} \stackrel{!}{=} 1$$

Recap Maximum Likelihood Estimation for Gaussian and Multinomial Distributions

Given N IID observations $D = \{x_1,\ldots,x_N\}$.

- For a **multivariate Gaussian** model $p(x_n|\theta) = \mathcal{N}(x_n|\mu, \Sigma)$, we obtain ML estimates

$$\hat{\mu} = \frac{1}{N}\sum_n x_n \qquad \text{(sample mean)}$$

$$\hat{\Sigma} = \frac{1}{N}\sum_n (x_n - \hat{\mu})(x_n - \hat{\mu})^T \qquad \text{(sample variance)}$$

- For discrete outcomes modeled by a 1-of-K **categorical distribution** we find

$$\hat{\mu}_k = \frac{1}{N}\sum_n x_{nk} \quad \left(= \frac{m_k}{N}\right)$$

$$\text{(sample proportion)}$$

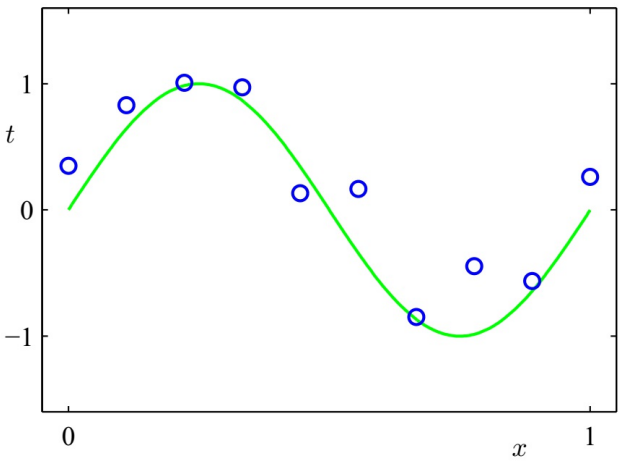
- Note the similarity for the means between discrete and continuous data.

Regression

Preliminaries

- Goal
  - Introduction to Bayesian (Linear) Regression
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 152-158
    - In this and forthcoming lectures, we will make use of some elementary matrix calculus. The most important formulas are summarized at the bottom of this notebook in an OPTIONAL SLIDE on matrix calculus. For derivations, see Bishop Appendix C.

Regression - Illustration



Given a set of (noisy) data measurements, find the 'best' relation between an input variable $x \in \mathbb{R}^M$ and input-dependent outcomes $y \in \mathbb{R}$.

Regression vs Density Estimation

- Observe N IID data **pairs** $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ with $x_n \in \mathbb{R}^M$ and $y_n \in \mathbb{R}$.

- Assume that we are interested in (a model for) the responses y_n for **given inputs** x_n?, I.o.w. we are interested in building a model for the conditional distribution $p(y|x)$.

- Note that, since $p(x, y) = p(y|x)\, p(x)$, building a model $p(y|x)$ is similar to density estimation with the assumption that x is drawn from a uniform distribution.

Bayesian Linear Regression

- Next, we discuss (1) model specification, (2) Inference and (3) a prediction application for a Bayesian linear regression problem.

1. Model Specification

- In a traditional *regression* model, we try to 'explain the data' by a purely deterministic function $f(x_n, w)$, plus a purely random term ϵ_n for 'unexplained noise':

$$y_n = f(x_n, w) + \epsilon_n$$

- In a *linear regression* model, i.e., linear w.r.t. the parameters w, we assume that

$$f(x_n, w) = \sum_{j=0}^{M-1} w_j \phi_j(x_n) = w^T \phi(x_n)$$

  where $\phi_j(x)$ are called basis functions.
  - For notational simplicity, from now on we will assume $f(x_n, w) = w^T x_n$, with $x_n \in \mathbb{R}^M$.

- In *ordinary linear regression* , the noise process ϵ_n is zero-mean Gaussian with constant variance, i.e.

$$\epsilon_n \sim \mathcal{N}(0, \beta^{-1})\,.$$

- Hence, given a data set $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, the likelihood for an ordinary linear regression model is

$$\begin{aligned} p(y \,|\, \mathbf{X}, w, \beta) &= \mathcal{N}(y \,|\, \mathbf{X}w, \beta^{-1}\mathbf{I}) \\ &= \prod_n \mathcal{N}(y_n \,|\, w^T x_n, \beta^{-1}) \end{aligned} \tag{B-3.10}$$

where $w = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_M \end{pmatrix}$, the $(N \times M)$-dim matrix $\mathbf{X} = \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_N^T \end{pmatrix} = \begin{pmatrix} x_{11}, x_{12}, \ldots, x_{1M} \\ x_{21}, x_{22}, \ldots, x_{2M} \\ \vdots \\ x_{N1}, x_{N2}, \ldots, x_{NM} \end{pmatrix}$ and $y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$.

- For full Bayesian learning we should also choose a prior $p(w)$:

$$p(w \,|\, \alpha) = \mathcal{N}(w \,|\, 0, \alpha^{-1}\mathbf{I}) \tag{B-3.52}$$

- For simplicity, we will assume that α and β are fixed and known.

2. Inference

- We'll do Bayesian inference for the parameters w.

$$\begin{aligned} p(w|D) &\propto p(D|w) \cdot p(w) \\ &= \mathcal{N}(y \,|\, \mathbf{X}w, \beta^{-1}\mathbf{I}) \cdot \mathcal{N}(w \,|\, 0, \alpha^{-1}\mathbf{I}) \\ &\propto \exp\Big(-\frac{\beta}{2}(y - \mathbf{X}w)^T(y - \mathbf{X}w) - \frac{\alpha}{2} w^T w\Big) \qquad \text{(B-3.55)} \\ &= \exp\Big(-\frac{1}{2} w^T \underbrace{(\beta \mathbf{X}^T\mathbf{X} + \alpha\mathbf{I})}_{\Lambda_N} w + \underbrace{(\beta \mathbf{X}^T y)}_{\eta_N}{}^T w \\ &\qquad - \frac{\beta}{2} y^T y\Big) \\ &\propto \mathcal{N}_c\left(w \,|\, \eta_N, \Lambda_N\right) \end{aligned}$$

with natural parameters (see the natural parameterization of Gaussian):

$$\eta_N = \beta \mathbf{X}^T y$$
$$\Lambda_N = \beta \mathbf{X}^T \mathbf{X} + \alpha \mathbf{I}$$

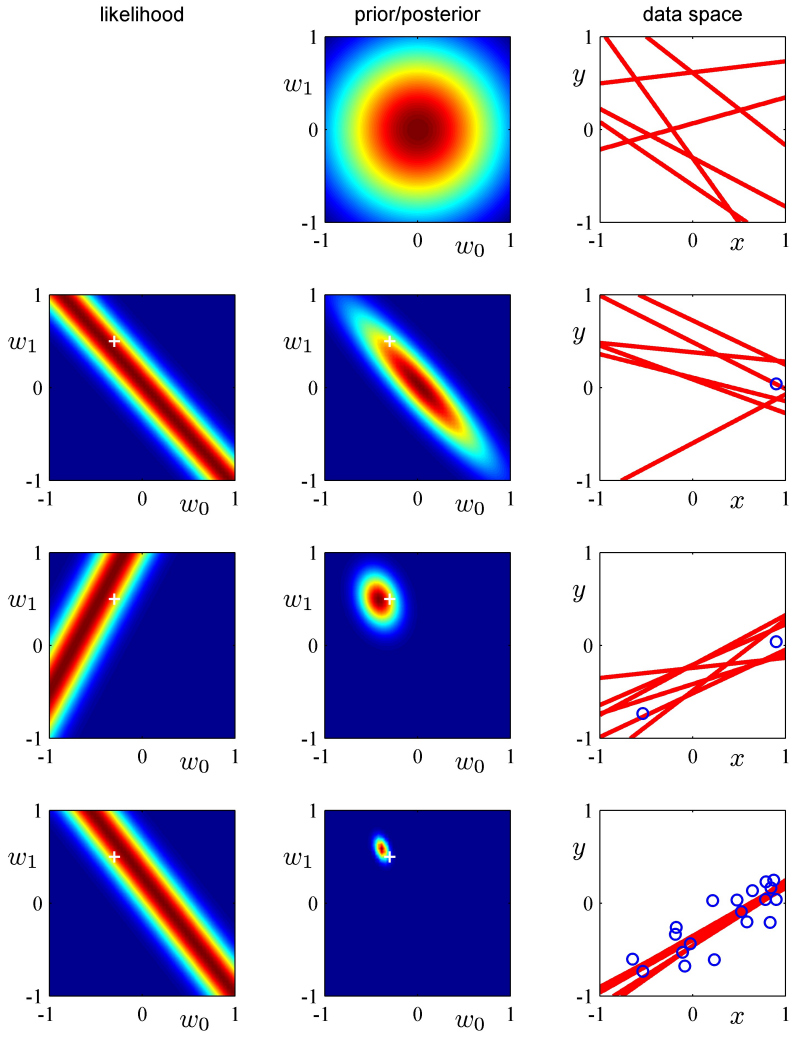
- Or equivalently (in the moment parameterization of the Gaussian):

$$p(w|D) = \mathcal{N}\left(w \,|\, m_N, S_N\right) \qquad \text{(B-3.49)}$$
$$m_N = \beta S_N \mathbf{X}^T y \qquad \text{(B-3.53)}$$
$$S_N = \left(\alpha \mathbf{I} + \beta \mathbf{X}^T \mathbf{X}\right)^{-1} \qquad \text{(B-3.54)}$$

- Note that B-3.53 and B-3.54 combine to

$$m_N = \left(\frac{\alpha}{\beta}\mathbf{I} + \mathbf{X}^T \mathbf{X}\right)^{-1} \mathbf{X}^T y \,.$$

- (Bishop Fig.3.7) Illustration of sequential Bayesian learning for a simple linear model of the form $y(x, w) = w_0 + w_1 x$. (Bishop Fig.3.7, detailed description at Bishop, pg.154.)



3. Application: Predictive Distribution

- Assume we are interested in the distribution $p(y_\bullet \mid x_\bullet, D)$ for a new input $x_\bullet$. This can be worked out as (exercise B-3.10)

$$
\begin{aligned}
p(y_\bullet \mid x_\bullet, D) &= \int p(y_\bullet \mid x_\bullet, w) p(w \mid D)\, \mathrm{d}w \\
&= \int \mathcal{N}(y_\bullet \mid w^T x_\bullet, \beta^{-1}) \mathcal{N}(w \mid m_N, S_N)\, \mathrm{d}w \\
&= \int \mathcal{N}(y_\bullet \mid z, \beta^{-1}) \underbrace{\mathcal{N}(z \mid x_\bullet^T m_N, x_\bullet^T S_N x_\bullet)\, \mathrm{d}z}_{= \mathcal{N}(w \mid m_N, S_N)\, \mathrm{d}w} \\
&\qquad\qquad (\text{sub. } z = x_\bullet^T w) \\
&= \int \underbrace{\underbrace{\mathcal{N}(z \mid y_\bullet, \beta^{-1})}_{\text{switch } z \text{ and } y_\bullet} \mathcal{N}(z \mid x_\bullet^T m_N, x_\bullet^T S_N x_\bullet)\, \mathrm{d}z}_{\text{Use Gaussian product formula SRG-6}} \\
&= \int \mathcal{N}\left(y_\bullet \mid m_N^T x_\bullet, \sigma_N^2(x_\bullet)\right) \underbrace{\mathcal{N}\left(z \mid \cdot, \cdot\right)}_{\text{integrate this out}}\, \mathrm{d}z \\
&= \mathcal{N}\left(y_\bullet \mid m_N^T x_\bullet, \sigma_N^2(x_\bullet)\right)
\end{aligned}
$$

with

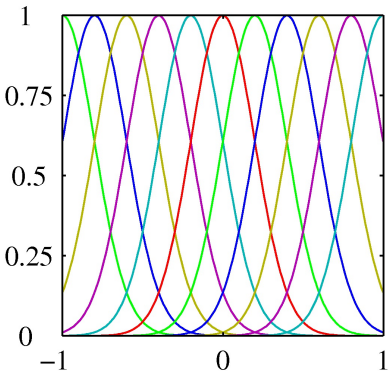
$$
\sigma_N^2(x_\bullet) = \beta^{-1} + x_\bullet^T S_N x_\bullet \tag{B-3.59}
$$

- So, the uncertainty $\sigma_N^2(x_\bullet)$ about the output $y_\bullet$ contains both uncertainty about the process (β^{-1}) and about the model parameters ($x_\bullet^T S_N x_\bullet$).
- (See the OPTIONAL SLIDE below for the step in this derivation where $\mathcal{N}(w \mid m_N, S_N)\, \mathrm{d}w$ gets replaced $\mathcal{N}(z \mid x_\bullet^T m_N, x_\bullet^T S_N x_\bullet)\, \mathrm{d}z$.)

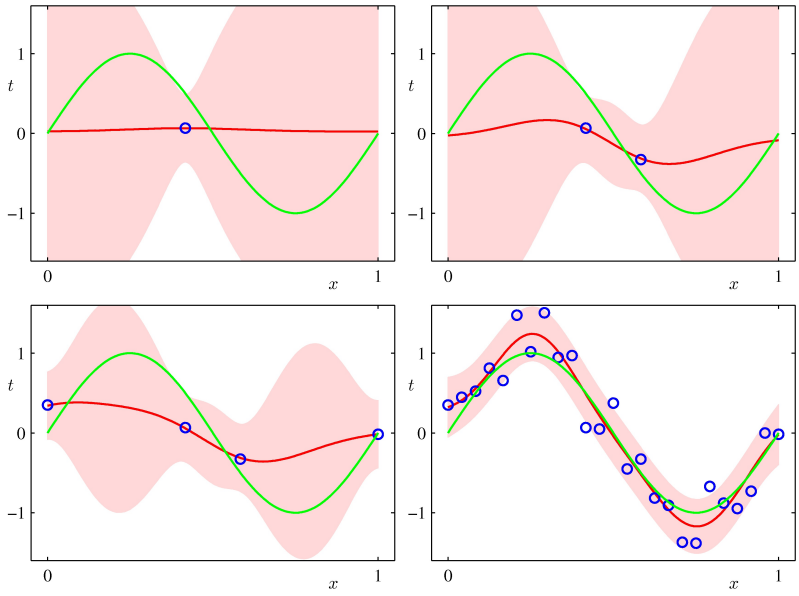
Example Predictive Distribution

- As an example, let's do Bayesian Linear Regression for a synthetic sinusoidal data set and a model with 9 Gaussian basis functions

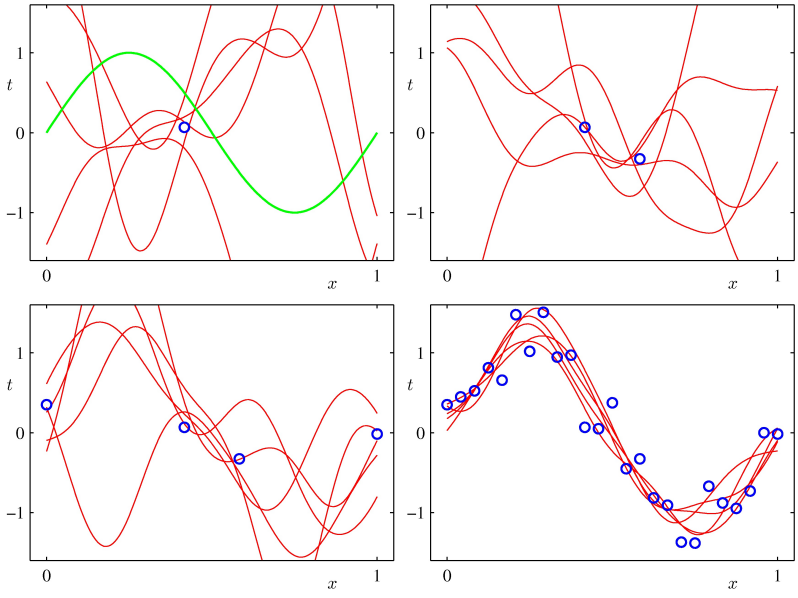
$$
\begin{aligned}
y_n &= \sum_{m=1}^{9} w_m \phi_m(x_n) + \epsilon_n \\
\phi_m(x_n) &= \exp\left(\frac{x_n - \mu_m}{\sigma^2}\right) \\
\epsilon_n &\sim \mathcal{N}(0, \beta^{-1})
\end{aligned}
$$



- The predictive distributions for y are shown in the following plots (Bishop, Fig.3.8)



- And some plots of draws of posteriors for the functions $w^T\phi(x)$ (Bishop, Fig.3.9)



Maximum Likelihood Estimation for Linear Regression Model

- Recall the posterior mean for the weight vector

$$m_N = \left(\frac{\alpha}{\beta}\mathbf{I} + \mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T y$$

  where α is the prior precision for the weights.

- The Maximum Likelihood solution for w is obtained by letting $\alpha \to 0$, which leads to

$$\hat{w}_{\text{ML}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T y$$

- The matrix $\mathbf{X}^\dagger \equiv (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ is also known as the **Moore-Penrose pseudo-inverse** (which is sort-of-an-inverse for non-square matrices).

- Note that if we have fewer training samples than input dimensions, i.e., if $N < M$, then $\mathbf{X}^T\mathbf{X}$ will not be invertible and maximum likelihood blows up. The Bayesian solution does not suffer from this problem.

Least-Squares Regression

- (You may say that) we don't need to work with probabilistic models. E.g., there's also the deterministic **least-squares** solution: minimize sum of squared errors,

$$\hat{w}_{\mathrm{LS}} = \arg\min_w \sum_n \left(y_n - w^T x_n\right)^2 = \arg\min_w \left(y - \mathbf{X}w\right)^T \left(y - \mathbf{X}w\right)$$

- Setting the gradient $\frac{\partial (y-\mathbf{X}w)^T (y-\mathbf{X}w)}{\partial w} = -2\mathbf{X}^T \left(y - \mathbf{X}w\right)$ to zero yields the so-called **normal equations** $\mathbf{X}^T\mathbf{X}\hat{w}_{\mathrm{LS}} = \mathbf{X}^T y$ and consequently

$$\hat{w}_{\mathrm{LS}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T y$$

which is the same answer as we got for the maximum likelihood weights $\hat{w}_{\mathrm{ML}}$.

- $\Rightarrow$ Least-squares regression ($\hat{w}_{\mathrm{LS}}$) corresponds to the (probabilistic) maximum likelihood solution ($\hat{w}_{\mathrm{ML}}$) if the probabilistic model includes the following assumptions:
  1. The observations are independently and identically distributed (**IID**) (this determines how errors are combined), and
  2. The noise signal $\epsilon_n \sim \mathcal{N}(0,\ \beta^{-1})$ is **Gaussian** distributed (determines the error metric)

- If you use the Least-Squares method, you cannot see (nor modify) these assumptions. The probabilistic method forces you to state all assumptions explicitly!

Not Identically Distributed Data

- Let's do an example regarding changing our assumptions. What if we assume that the variance of the measurement error varies with the sampling index, $\epsilon_n \sim \mathcal{N}(0, \beta_n^{-1})$?

- The likelihood is now (using $\Lambda \triangleq \mathrm{diag}(\beta_n)$)

$$p(y \,|\, \mathbf{X}, w, \Lambda) = \mathcal{N}(y \,|\, \mathbf{X}w, \Lambda^{-1})\,.$$

- Combining this likelihood with the prior $p(w) = \mathcal{N}(w \,|\, 0, \alpha^{-1}\mathbf{I})$ leads to a posterior

$$\begin{aligned} p(w|D) &\propto p(D|w) \cdot p(w) \\ &= \mathcal{N}(y \,|\, \mathbf{X}w, \Lambda^{-1}\mathbf{I}) \cdot \mathcal{N}(w \,|\, 0, \alpha^{-1}\mathbf{I}) \\ &\propto \exp\left\{ \frac{1}{2}(y - \mathbf{X}w)^T \Lambda \left(y - \mathbf{X}w\right) + \frac{\alpha}{2} w^T w \right\} \\ &\propto \mathcal{N}\left(w \,|\, m_N, S_N\right) \end{aligned}$$

with

$$\begin{aligned} m_N &= S_N \mathbf{X}^T \Lambda y \\ S_N &= \left(\alpha\mathbf{I} + \mathbf{X}^T\Lambda\mathbf{X}\right)^{-1} \end{aligned}$$

- And maximum likelihood solution

$$\hat{w}_{\mathrm{ML}} = m_N|_{\alpha\to 0} = \left(\mathbf{X}^T\Lambda\mathbf{X}\right)^{-1}\mathbf{X}^T\Lambda y$$

- This maximum likelihood solution is also called the **Weighted Least Squares** (WLS) solution. (Note that we just stumbled upon it, the crucial aspect is appropriate model specification!)

- Note also that the dimension of Λ grows with the number of data points. In general, models for which the number of parameters grow as the number of observations increase are called **non-parametric models**.

Code Example: Least Squares vs Weighted Least Squares

- We'll compare the Least Squares and Weighted Least Squares solutions for a simple linear regression model with input-dependent noise:

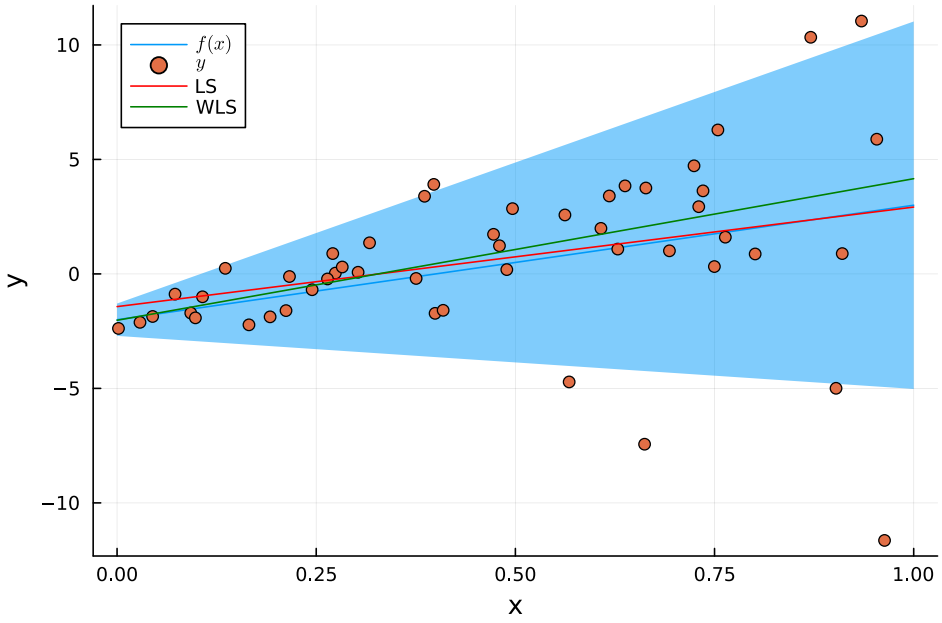
$$
\begin{aligned}
x &\sim \mathrm{Unif}[0,1] \\
y|x &\sim \mathcal{N}(f(x), v(x)) \\
f(x) &= 5x - 2 \\
v(x) &= 10e^{2x^2} - 9.5 \\
\mathcal{D} &= \{(x_1, y_1), \ldots, (x_N, y_N)\}
\end{aligned}
$$

In [1]:
```
using Pkg; Pkg.activate("../."); Pkg.instantiate();
using IJulia; try IJulia.clear_output(); catch _ end
```

Out[1]:0

In [2]:
```
using Plots, Distributions, LaTeXStrings, LinearAlgebra

# Model specification: y|x ~ 𝒩(f(x), v(x))
f(x) = 5*x .- 2
v(x) = 10*exp.(2*x.^2) .- 9.5 # input dependent noise variance
x_test = [0.0, 1.0]
plot(x_test, f(x_test), ribbon=sqrt.(v(x_test)), label=L"f(x)") # plot f(x)

# Generate N samples (x,y), where x ~ Unif[0,1]
N = 50
x = rand(N)
y = f(x) + sqrt.(v(x)) .* randn(N)
scatter!(x, y, xlabel="x", ylabel="y", label=L"y") # Plot samples

# Add constant to input so we can estimate both the offset and the slope
_x = [x ones(N)]
_x_test = hcat(x_test, ones(2))

# LS regression
w_ls = pinv(_x) * y
plot!(x_test, _x_test*w_ls, color=:red, label="LS") # plot LS solution

# Weighted LS regression
W = Diagonal(1 ./ v(x)) # weight matrix
w_wls = inv(_x'*W*_x) * _x' * W * y
plot!(x_test, _x_test*w_wls, color=:green, label="WLS") # plot WLS solution
```

Out[2]:



Some Final Remarks

- Aside from the above example, we also recommend that you read through the RxInfer Bayesian Linear Regression example.
- In this lesson, we focussed on modelling the map from given inputs x to uncertain outputs y, or more formally, on the distribution $p(y|x)$. What if you want to fit the best curve through a data set $\{(x_1, y_1), \ldots, (x_N, y_N)\}$ where both variables x_n and y_n are subject to errors? In other words, we must now also fit a model $p(x)$ for the inputs, leading to a generative model $p(y, x) = p(y|x)p(x)$.

- While this is a very common problem that occurs throughout the sciences, a proper solution to this problem is still hardly covered in statistics textbooks. Edwin Jaynes discusses a fully Bayesian solution in his 1990 paper on Straight Line Fitting - A Bayesian Solution. (Optional reading).

OPTIONAL SLIDES

Some Useful Matrix Calculus

When doing derivatives with matrices, e.g. for maximum likelihood estimation, it will be helpful to be familiar with some matrix calculus. We shortly recapitulate used formulas here.

- We define the **gradient** of a scalar function $f(A)$ w.r.t. an $n \times k$ matrix A as

$$\nabla_A f \triangleq \begin{bmatrix} \frac{\partial f}{\partial a_{11}} & \frac{\partial f}{\partial a_{12}} & \cdots & \frac{\partial f}{\partial a_{1k}} \\ \frac{\partial f}{\partial a_{21}} & \frac{\partial f}{\partial a_{22}} & \cdots & \frac{\partial f}{\partial a_{2k}} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial f}{\partial a_{n1}} & \frac{\partial f}{\partial a_{n2}} & \cdots & \frac{\partial f}{\partial a_{nk}} \end{bmatrix}$$

- The following formulas are useful (see Bishop App.-C)

$$\begin{aligned} |A^{-1}| &= |A|^{-1} && \text{(B-C.4)} \\ \nabla_A \log|A| &= (A^T)^{-1} = (A^{-1})^T && \text{(B-C.28)} \\ \mathrm{Tr}[ABC] &= \mathrm{Tr}[CAB] = \mathrm{Tr}[BCA] && \text{(B-C.9)} \\ \nabla_A \mathrm{Tr}[AB] &= \nabla_A \mathrm{Tr}[BA] = B^T && \text{(B-C.25)} \\ \nabla_A \mathrm{Tr}[ABA^T] &= A(B+B^T) && \text{(B-C.27)} \\ \nabla_x x^T A x &= (A + A^T)x && \text{(from B-C.27)} \\ \nabla_X a^T X b &= \nabla_X \mathrm{Tr}[b a^T X] = a b^T \end{aligned}$$

Derivation Predictive Distribution

- In the derivation of the predictive distribution, we replaced $\mathcal{N}(w \,|\, m_N, S_N)\,\mathrm{d}w$ with $\mathcal{N}(z \,|\, x_\bullet^T m_N, x_\bullet^T S_N x_\bullet)\,\mathrm{d}z$. Here we discuss why that is allowed.
- Since $z = x^T w$ (drop the bullet for notational simplicity), we have

$$p(z) = \mathcal{N}(z|m_z, \Sigma_z)$$

with

$$\begin{aligned} m_z &:= E[z] = E[x^T w] = x^T E[w] = x^T m_N \\ \Sigma_z &:= E[(z - m_z)(z - m_z)^T] \\ &= E[(x^T w - x^T m_N)(x^T w - x^T m_N)^T] \\ &= x^T E[(w - m_N)(w - m_N)^T] x \\ &= x^T S_N x \end{aligned}$$

- Then we equate probability masses in both domains:

$$\mathcal{N}(z|m_z, \Sigma_z)\mathrm{d}z = \mathcal{N}(w|m_N, S_N)\mathrm{d}w$$

$$\Rightarrow \mathcal{N}(z|x^T m_N, x^T S_N x)\mathrm{d}z = \mathcal{N}(w|m_N, S_N)\mathrm{d}w$$

Generative Classification

Preliminaries

- Goal
  - Introduction to linear generative classification with a Gaussian-categorical generative model
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 196-202 (section 4.2 focusses on binary classification, whereas in these lecture notes we describe generative classification for multiple classes).

Challenge: an apple or a peach?

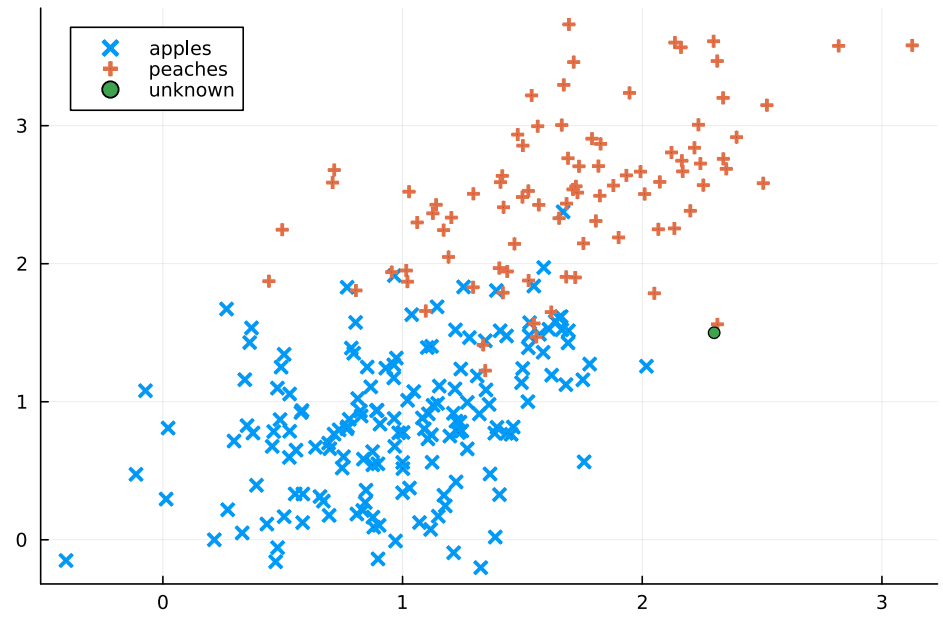
- **Problem**: You're given numerical values for the skin features roughness and color for 200 pieces of fruit, where for each piece of fruit you also know if it is an apple or a peach. Now you receive the roughness and color values for a new piece of fruit but you don't get its class label (apple or peach). What is the probability that the new piece is an apple?
- **Solution**: To be solved later in this lesson.
- Let's first generate a data set (see next slide).

In [1]:
```
using Pkg; Pkg.activate("../."); Pkg.instantiate();
using IJulia; try IJulia.clear_output(); catch _ end
```

Out[1]:0

In [2]:
```
using Plots, Distributions

N = 250; p_apple = 0.7; Σ = [0.2 0.1; 0.1 0.3]
p_given_apple = MvNormal([1.0, 1.0], Σ)                          # p(X|y=apple)
p_given_peach = MvNormal([1.7, 2.5], Σ)                          # p(X|y=peach)
X = Matrix{Float64}(undef,2,N); y = Vector{Bool}(undef,N)        # true corresponds to apple
for n=1:N
    y[n] = (rand() < p_apple)                                    # Apple or peach?
    X[:,n] = y[n] ? rand(p_given_apple) : rand(p_given_peach)    # Sample features
end
X_apples = X[:,findall(y)]'; X_peaches = X[:,findall(.!y)]'      # Sort features on class
x_test = [2.3; 1.5]                                              # Features of 'new' data point

scatter(X_apples[:,1], X_apples[:,2], label="apples", marker=:x, markerstrokewidth=3)        # apples
scatter!(X_peaches[:,1], X_peaches[:,2], label="peaches", marker=:+,  markerstrokewidth=3)   # peaches
scatter!([x_test[1]], [x_test[2]], label="unknown")                                          # 'new' unlabell
```

Out[2]:



Generative Classification Problem Statement

- Given is a data set $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$
  - inputs $x_n \in \mathbb{R}^M$ are called **features**.

- outputs $y_n \in \mathcal{C}_k$, with $k = 1, \ldots, K$; The **discrete** targets $\mathcal{C}_k$ are called **classes**.

- We will again use the 1-of-K notation for the discrete classes. Define the binary **class selection variable**

$$y_{nk} = \begin{cases} 1 & \text{if } y_n \in \mathcal{C}_k \\ 0 & \text{otherwise} \end{cases}$$

  - (Hence, the notations $y_{nk} = 1$ and $y_n \in \mathcal{C}_k$ mean the same thing.)

- The plan for generative classification: build a model for the joint pdf $p(x, y) = p(x|y)p(y)$ and use Bayes to infer the posterior class probabilities

$$p(y|x) = \frac{p(x|y)p(y)}{\sum_{y'} p(x|y')p(y')} \propto p(x|y)\, p(y)$$

1 - Model specification

Likelihood

- Assume Gaussian **class-conditional distributions** with **equal covariance matrix** across the classes,

$$p(x_n|\mathcal{C}_k) = \mathcal{N}(x_n|\mu_k, \Sigma)$$

  with notational shorthand: $\mathcal{C}_k \triangleq (y_n \in \mathcal{C}_k)$.

Prior

- We use a categorical distribution for the class labels y_{nk}:

$$p(\mathcal{C}_k) = \pi_k$$

- Hence, using the one-hot coding formulation for y_{nk}, the generative model $p(x_n, y_n)$ can be written as

$$\begin{aligned} p(x_n, y_n) &= \prod_{k=1}^{K} p(x_n, y_{nk} = 1)^{y_{nk}} \\ &= \prod_{k=1}^{K} \left(\pi_k \cdot \mathcal{N}(x_n|\mu_k, \Sigma)\right)^{y_{nk}} \end{aligned}$$

- We will refer to this model as the **Gaussian-Categorical Model** (GCM).
  - N.B. In the literature, this model (with possibly unequal Σ_k across classes) is often called the Gaussian Discriminant Analysis model and the special case with equal covariance matrices $\Sigma_k = \Sigma$ is also called Linear Discriminant Analysis. We think these names are a bit unfortunate as it may lead to confusion with the discriminative method for classification.

- As usual, once the model has been specified, the rest (inference for parameters and model prediction) through straight probability theory.

Computing the log-likelihood

- The log-likelihood given the full data set $D = \{(x_n, y_n), n = 1, 2, \ldots, N\}$ is then

$$\begin{aligned}
\log\, p(D|\theta) &\overset{\text{IID}}{=} \sum_n \log \prod_k p(x_n, y_{nk}=1\,|\,\theta)^{y_{nk}} \\
&= \sum_{n,k} y_{nk} \log p(x_n, y_{nk}=1\,|\,\theta) \\
&= \sum_{n,k} y_{nk} \log p(x_n|y_{nk}=1) + \sum_{n,k} y_{nk} \log \\
&\qquad\qquad p(y_{nk}=1) \\
&= \sum_{n,k} y_{nk} \log \mathcal{N}(x_n|\mu_k, \Sigma) + \sum_{n,k} y_{nk} \log \pi_k \\
&= \sum_{n,k} y_{nk} \underbrace{\log \mathcal{N}(x_n|\mu_k, \Sigma)}_{\text{see Gaussian lecture}} \\
&\quad + \underbrace{\sum_k m_k \log \pi_k}_{\text{see multinomial lecture}}
\end{aligned}$$

where we used $m_k \triangleq \sum_n y_{nk}$.

2 - Parameter Inference for Classification

- We'll do Maximum Likelihood estimation for $\theta = \{\pi_k, \mu_k, \Sigma\}$ from data D.
- Recall (from the previous slide) the log-likelihood (LLH)

$$\begin{aligned}
\log\, p(D|\theta) &= \sum_{n,k} y_{nk} \underbrace{\log \mathcal{N}(x_n|\mu_k, \Sigma)}_{\text{Gaussian}} \\
&\quad + \underbrace{\sum_k m_k \log \pi_k}_{\text{multinomial}}
\end{aligned}$$

- Maximization of the LLH for the GDA model breaks down into
  - **Gaussian density estimation** for parameters μ_k, Σ, since the first term contains exactly the log-likelihood for MVG density estimation. We've already done this, see the Gaussian distribution lesson.
  - **Multinomial density estimation** for class priors π_k, since the second term holds exactly the log-likelihood for multinomial density estimation, see the Multinomial distribution lesson.
- The ML for multinomial class prior (we've done this before!)

$$\hat{\pi}_k = \frac{m_k}{N}$$

- Now group the data into separate classes and do MVG ML estimation for class-conditional parameters (we've done this before as well):

$$\begin{aligned}
\hat{\mu}_k &= \frac{\sum_n y_{nk} x_n}{\sum_n y_{nk}} = \frac{1}{m_k} \sum_n y_{nk} x_n \\
\hat{\Sigma} &= \frac{1}{N} \sum_{n,k} y_{nk} (x_n - \hat{\mu}_k)(x_n - \hat{\mu}_k)^T \\
&\qquad\qquad = \sum_k \hat{\pi}_k \\
&\qquad\qquad \cdot \underbrace{\left(\frac{1}{m_k} \sum_n y_{nk} (x_n - \hat{\mu}_k)(x_n - \hat{\mu}_k)^T \right)}_{\text{class-cond. variance}} \\
&= \sum_k \hat{\pi}_k \cdot \hat{\Sigma}_k
\end{aligned}$$

where $\hat{\pi}_k$, $\hat{\mu}_k$ and $\hat{\Sigma}_k$ are the sample proportion, sample mean and sample variance for the kth class, respectively.

- Note that the binary class selection variable y_{nk} groups data from the same class.

3 - Application: Class prediction for new Data

- Let's apply the trained model to predict the class for given a 'new' input $x_\bullet$:

$$
\begin{aligned}
p(\mathcal{C}_k|x_\bullet, D) &= \int p(\mathcal{C}_k|x_\bullet, \theta)\ \underbrace{p(\theta|D)}_{\text{ML: } \delta(\theta-\hat{\theta})}\ \mathrm{d}\theta \\
&= p(\mathcal{C}_k|x_\bullet, \hat{\theta}) \\
&\propto p(\mathcal{C}_k)\, p(x_\bullet|\mathcal{C}_k) \\
&= \hat{\pi}_k \cdot \mathcal{N}(x_\bullet|\hat{\mu}_k, \hat{\Sigma}) \\
&\propto \hat{\pi}_k \exp\left\{ -\frac{1}{2}(x_\bullet - \hat{\mu}_k)^T \hat{\Sigma}^{-1} (x_\bullet - \hat{\mu}_k) \right\} \\
&= \exp \\
&\quad \Big\{ \underbrace{-\frac{1}{2} x_\bullet^T \hat{\Sigma}^{-1} x_\bullet}_{\text{not a function of } k} \\
&\quad + \underbrace{\hat{\mu}_k^T \hat{\Sigma}^{-1}}_{\beta_k^T} x_\bullet \underbrace{- \frac{1}{2} \hat{\mu}_k^T \hat{\Sigma}^{-1} \hat{\mu}_k + \log \hat{\pi}_k}_{\gamma_k} \Big\} \\
&\propto \frac{1}{Z} \exp\{\beta_k^T x_\bullet + \gamma_k\} \\
&\triangleq \sigma\left(\beta_k^T x_\bullet + \gamma_k\right)
\end{aligned}
$$

where $\sigma(a_k) \triangleq \frac{\exp(a_k)}{\sum_{k'} \exp(a_{k'})}$ is called a **softmax** (a.k.a. **normalized exponential**) function, and

$$
\begin{aligned}
\beta_k &= \hat{\Sigma}^{-1} \hat{\mu}_k \\
\gamma_k &= -\frac{1}{2} \hat{\mu}_k^T \hat{\Sigma}^{-1} \hat{\mu}_k + \log \hat{\pi}_k \\
Z &= \sum_{k'} \exp\{\beta_{k'}^T x_\bullet + \gamma_{k'}\}\,. \\
&\quad \text{(normalization constant)}
\end{aligned}
$$

- The softmax function is a smooth approximation to the max-function. Note that we did not a priori specify a softmax posterior, but rather it followed from applying Bayes rule to the prior and likelihood assumptions.

- Note the following properties of the softmax function $\sigma(a_k)$:
  - $\sigma(a_k)$ is monotonicaly ascending function and hence it preserves the order of a_k. That is, if $a_j > a_k$ then $\sigma(a_j) > \sigma(a_k)$.
  - $\sigma(a_k)$ is always a proper probability distribution, since $\sigma(a_k) > 0$ and $\sum_k \sigma(a_k) = 1$.

Discrimination Boundaries

- The class log-posterior $\log p(\mathcal{C}_k|x) \propto \beta_k^T x + \gamma_k$ is a linear function of the input features.

- Thus, the contours of equal probability (**discriminant functions**) are lines (hyperplanes) in the feature space

$$
\log \frac{p(\mathcal{C}_k|x, \theta)}{p(\mathcal{C}_j|x, \theta)} = \beta_{kj}^T x + \gamma_{kj} = 0
$$

where we defined $\beta_{kj} \triangleq \beta_k - \beta_j$ and similarly for γ_{kj}.

- How to classify a new input $x_\bullet$? The Bayesian answer is a posterior distribution $p(\mathcal{C}_k|x_\bullet)$. If you must choose, then the class with

maximum posterior class probability

$$\begin{aligned} k^* &= \arg\max_k p(\mathcal{C}_k|x_\bullet) \\ &= \arg\max_k \left(\beta_k^T x_\bullet + \gamma_k\right) \end{aligned}$$

is an appealing decision.

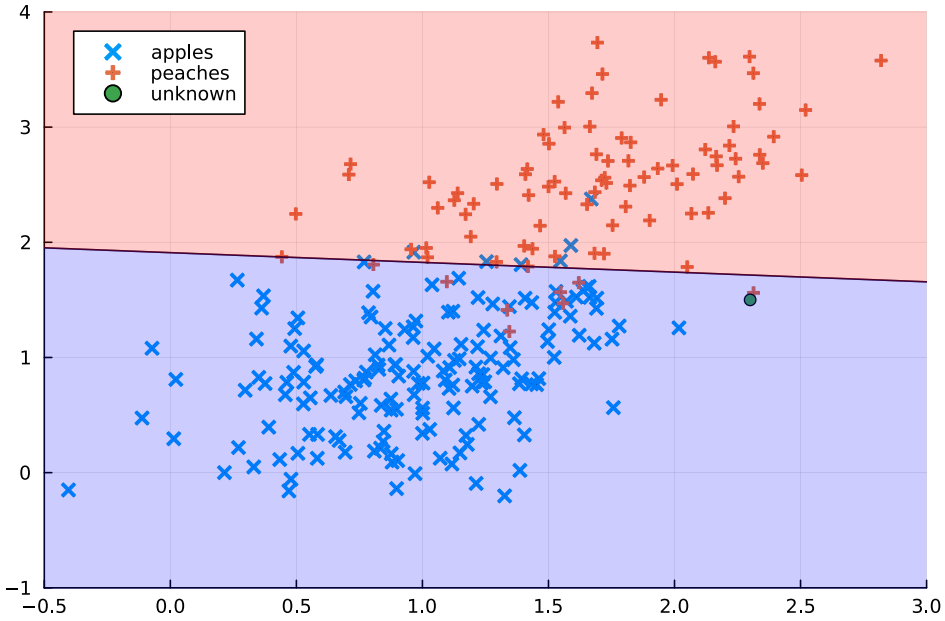
Code Example: Working out the "apple or peach" example problem

We'll apply the above results to solve the "apple or peach" example problem.

In [3]:
```
# Make sure you run the data-generating code cell first
using Distributions, Plots
# Multinomial (in this case binomial) density estimation
p_apple_est = sum(y.==true) / length(y)
π_hat = [p_apple_est; 1-p_apple_est]

# Estimate class-conditional multivariate Gaussian densities
d1 = fit_mle(FullNormal, X_apples')  # MLE density estimation d1 = N(μ₁, Σ₁)
d2 = fit_mle(FullNormal, X_peaches') # MLE density estimation d2 = N(μ₂, Σ₂)
Σ = π_hat[1]*cov(d1) + π_hat[2]*cov(d2) # Combine Σ₁ and Σ₂ into Σ
conditionals = [MvNormal(mean(d1), Σ); MvNormal(mean(d2), Σ)] # p(x|C)

# Calculate posterior class probability of x· (prediction)
function predict_class(k, X) # calculate p(Ck|X)
    norm = π_hat[1]*pdf(conditionals[1],X) + π_hat[2]*pdf(conditionals[2],X)
    return π_hat[k]*pdf(conditionals[k], X) ./ norm
end
println("p(apple|x=x·) = $(predict_class(1,x_test))")

# Discrimination boundary of the posterior (p(apple|x;D) = p(peach|x;D) = 0.5)
β(k) = inv(Σ)*mean(conditionals[k])
γ(k) = -0.5 * mean(conditionals[k])' * inv(Σ) * mean(conditionals[k]) + log(π_hat[k])
function discriminant_x2(x1)
    # Solve discriminant equation for x2
    β12 = β(1) .- β(2)
    γ12 = (γ(1) .- γ(2))[1,1]
    return -1*(β12[1]*x1 .+ γ12) ./ β12[2]
end

scatter(X_apples[:,1], X_apples[:,2], label="apples", marker=:x, markerstrokewidth=3)    # apples
scatter!(X_peaches[:,1], X_peaches[:,2], label="peaches", marker=:+,  markerstrokewidth=3) # peaches
scatter!([x_test[1]], [x_test[2]], label="unknown")            # 'new' unlabelled data point

x1 = range(-1,length=10,stop=3)
plot!(x1, discriminant_x2(x1), color="black", label="") # Plot discrimination boundary
plot!(x1, discriminant_x2(x1), fillrange=-10, alpha=0.2, color=:blue, label="")
plot!(x1, discriminant_x2(x1), fillrange=10, alpha=0.2, color=:red, xlims=(-0.5, 3), ylims=(-1, 4), label=""
```

```
p(apple|x=x·) = 0.7696297495358562
```

Out[3]:



Why Be Bayesian?

- A student in one of the previous years posed the following question at Piazza:

  > " After re-reading topics regarding generative classification, this question popped into my mind: Besides the sole purpose of the lecture, which is getting to know the concepts of generative classification and how to implement them, are there any advantages of using this instead of using deep neural nets? DNNs seem simpler and more powerful."

- The following answer was provided:

  - If you are only are interested in approximating a function, say $y = f_\theta(x)$, and you have lots of examples $\{(x_i, y_i)\}$ of desired behavior, then often a non-probabilistic DNN is a fine approach.
  - However, if you are willing to formulate your models in a probabilistic framework, you can improve on the deterministic approach in many ways, eg,

    1. Bayesian evidence for model performance assessment. This means you can use the whole data set for training without an ad-hoc split into testing and training data sets.
    2. Uncertainty about parameters in the model is a measure that allows you to do *active learning*, ie, choose data that is most informative (see also the lesson on intelligent agents). This will allow you to train on small data sets, whereas the deterministic DNNs generally require much larger data sets.
    3. Prediction with uncertainty/confidence bounds.
    4. Explicit specification and separation of your assumptions.
    5. A framework that supports scoring both accuracy and model complexity in the same currency (probability). How are you going to penalize the size of your network in a deterministic framework?
    6. Automatic learning rates, no tuning parameters. For instance, the Kalman gain is a data-dependent, optimal learning rate. How will *you* choose your learning rates in a deterministic framework? Trial and error?
    7. Principled absorption of different sources of knowledge. Eg, outcome of one set of experiments can be captured by a posterior distribution that serves as a prior distribution for the next set of experiments.

  Admittedly, it's not easy to understand the probabilistic approach, but it is worth the effort.

Recap Generative Classification

- Gaussian-Categorical Model specification:

$$p(x, \mathcal{C}_k | \, \theta) = \pi_k \cdot \mathcal{N}(x | \mu_k, \Sigma)$$

- If the class-conditional distributions are Gaussian with equal covariance matrices across classes ($\Sigma_k = \Sigma$), then the discriminant functions

are hyperplanes in feature space.

- ML estimation for $\{\pi_k, \mu_k, \Sigma\}$ in the GCM model breaks down to simple density estimation for Gaussian and multinomial/categorical distributions.

- Posterior class probability is a softmax function

$$p(\mathcal{C}_k|x,\theta) \propto \exp\{\beta_k^T x + \gamma_k\}$$

where $\beta_k = \Sigma^{-1}\mu_k$ and $\gamma_k = -\frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \log \pi_k$.

Discriminative Classification

Preliminaries

- Goal
  - Introduction to discriminative classification models
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp. 213 - 217 (Laplace approximation)
    - Bishop pp. 217 - 220 (Bayesian logistic regression)
    - T. Minka (2005), Discriminative models, not discriminative training

Challenge: difficult class-conditional data distributions

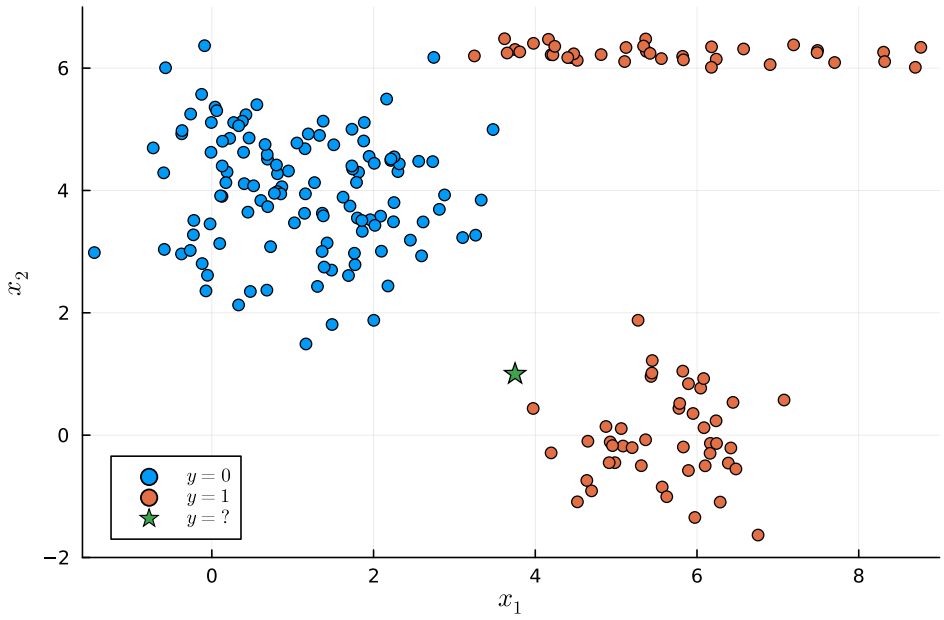
Our task will be the same as in the preceding class on (generative) classification. But this time, the class-conditional data distributions look very non-Gaussian, yet the linear discriminative boundary looks easy enough:

In [1]:
```
using Pkg; Pkg.activate("../."); Pkg.instantiate();
using IJulia; try IJulia.clear_output(); catch _ end
```

Out[1]:0

In [2]:
```
using Random; Random.seed!(1234);
# Generate dataset {(x1,y1),...,(xN,yN)}
# x is a 2-d feature vector [x_1;x_2]
# y ∈ {false,true} is a binary class label
# p(x|y) is multi-modal (mixture of uniform and Gaussian distributions)
using Plots, LaTeXStrings
include("./scripts/lesson8_helpers.jl")
N = 200
X, y = genDataset(N) # Generate data set, collect in matrix X and vector y
X_c1 = X[:,findall(.!y)]'; X_c2 = X[:,findall(y)]' # Split X based on class label
X_test = [3.75; 1.0] # Features of 'new' data point
function plotDataSet()
    result = scatter(X_c1[:,1], X_c1[:,2],markersize=4, label=L"y=0", xlabel=L"x_1", ylabel=L"x_2", xlims=(-
    scatter!(X_c2[:,1], X_c2[:,2],markersize=4, label=L"y=1")
    scatter!([X_test[1]], [X_test[2]], markersize=7, marker=:star, label=L"y=?")
    return result
end
plotDataSet()
```

Out[2]:



Main Idea of Discriminative Classification

- Again, a data set is given by $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ with $x_n \in \mathbb{R}^M$ and $y_n \in \mathcal{C}_k$, with $k = 1, \ldots, K$.
- Sometimes, the precise assumptions of the (Gaussian-Categorical) generative model

  $$p(x_n, y_n \in \mathcal{C}_k | \theta) = \pi_k \cdot \mathcal{N}(x_n | \mu_k, \Sigma)$$

  clearly do not match the data distribution.
- Here's an **IDEA**! Let's model the posterior

  $$p(y_n \in \mathcal{C}_k | x_n)$$

  directly, without any assumptions on the class densities.
- Similarly to regression, we will assume that the inputs x are given, so we wil not add a model $p(x)$ for input uncertainties.

Model Specification for Bayesian Logistic Regression

- We will work this idea out for a 2-class problem. Assume a data set is given by $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ with $x_n \in \mathbb{R}^M$ and $y_n \in \{0, 1\}$.
- What model should we use for the posterior distribution $p(y_n \in \mathcal{C}_k | x_n)$?

Likelihood

- We will take inspiration from the generative classification approach, where we derived the class posterior

  $$p(y_{nk} = 1 \,|\, x_n, \beta_k, \gamma_k) = \sigma(\beta_k^T x_n + \gamma_k)$$

  as a **softmax** function of a linear map of the input.
- Here, in logistic regression, we *choose* the 2-class softmax function (which is called the **logistic** function) with linear discrimination bounderies for the posterior class probability:

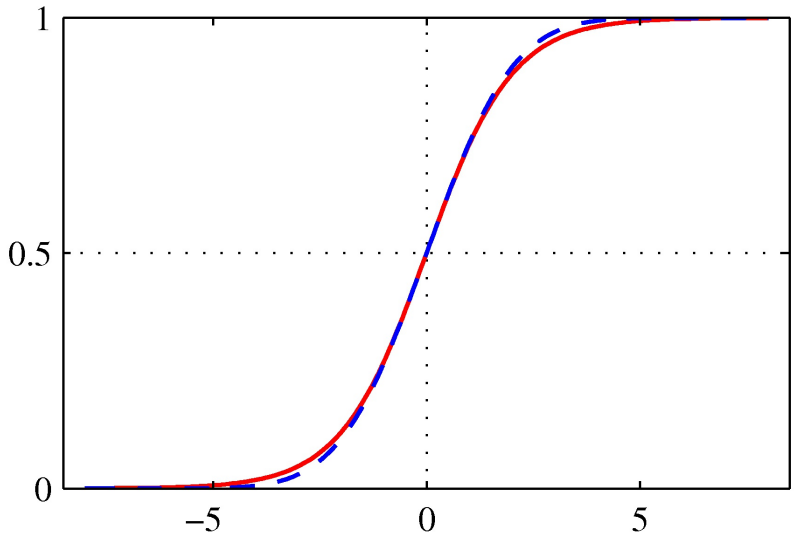
  $$p(y_n = 1 \,|\, x_n, w) = \sigma(w^T x_n)\,.$$

  where

  $$\sigma(a) = \frac{1}{1 + e^{-a}}$$

  is the *logistic* function.

- Clearly, it follows from this assumption that $p(y_n = 0 \,|\, x_n, w) = 1 - \sigma(w^T x_n)$.



- (Bishop fig.4.9). The logistic function $\sigma(a) = 1/(1 + e^{-a})$ (red), together with the scaled probit function $\Phi(\lambda a)$, for $\lambda^2 = \pi/8$ (in blue). We will use this approximation later in the Laplace approximation.

- Adding the other class ($y_n = 0$) leads to the following posterior class distribution:

$$\begin{aligned} p(y_n \,|\, x_n, w) &= \text{Bernoulli}\left(y_n \,|\, \sigma(w^T x_n)\right) \\ &= \sigma(w^T x_n)^{y_n} \left(1 - \sigma(w^T x_n)\right)^{(1-y_n)} \\ &= \sigma\left((2y_n - 1) w^T x_n\right) \end{aligned} \tag{B-4.89}$$

  - Note that for the 3rd equality, we have made use of the fact that $\sigma(-a) = 1 - \sigma(a)$.
  - Each of these three models in B-4.89 are **equivalent**. We mention all three notational options since they all appear in the literature.

- For the data set $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, the **likelihood function** for the parameters w is then given by

$$p(D|w) = \prod_{n=1}^{N} \sigma\left((2y_n - 1) w^T x_n\right)$$

- This choice for the class posterior is called **logistic regression**, in analogy to **linear regression**:

$$\begin{aligned} p(y_n|x_n, w) &= \mathcal{N}(y_n|w^T x_n, \beta^{-1}) && \text{for linear regression} \\ p(y_n|x_n, w) &= \sigma\left((2y_n - 1) w^T x_n\right) && \text{for logistic regression} \end{aligned}$$

- In the discriminative approach, the parameters w are **not** structured into $\{\mu, \Sigma, \pi\}$. In principle they are "free" parameters for which we can choose any value that seems appropriate. This provides discriminative approach with more flexibility than the generative approach.

Prior

- In *Bayesian* logistic regression, we often add a **Gaussian prior on the weights**:

$$p(w) = \mathcal{N}(w \,|\, m_0, S_0) \tag{B-4.140}$$

Some Notes on the Model

- Note that for generative classification, for the sake of simplicity, we used maximum likelihood estimation for the model parameters. In this lesson on discriminative classification, we specify both a prior and likelihood function for the parameters w, which allows us to compute a Bayesian posterior for the weights. In principle, we could have used Bayesian parameter estimation for the generative classification model as well (but the math is not suited for a introductory lesson).

- In the optional paper by T. Minka (2005), you can read how the model assumptions for discriminative classification can be re-interpreted as

a special generative model (this paper not for exam).

- As an exercise, please check that for logistic regression with $p(y_n = 1 \mid x_n, w) = \sigma(w^T x_n)$, the **discrimination boundary**, which can be computed by

$$\frac{p(y_n \in \mathcal{C}_1 | x_n)}{p(y_n \in \mathcal{C}_0 | x_n)} \stackrel{!}{=} 1$$

is a straight line, see Exercises.

Parameter Inference

- After model specification, the rest follows by application of probability theory.
- The posterior for the weights follows by Bayes rule

$$\underbrace{p(w \mid D)}_{\text{posterior}} = \frac{p(w)p(D|w)}{\int p(w)p(D|w)\mathrm{d}w}$$

$$= \frac{\overbrace{\mathcal{N}(w \mid m_0, S_0)}^{\text{prior}} \cdot \overbrace{\prod_{n=1}^N \sigma\left((2y_n - 1)w^T x_n\right)}^{\text{likelihood}}}{\underbrace{\int \mathcal{N}(w \mid m_0, S_0) \prod_{n=1}^N \sigma\left((2y_n - 1)w^T x_n\right) \mathrm{d}w}_{\text{evidence}}} \tag{B-4.142}$$

- In principle, Bayesian inference is done now!
- Unfortunately, the posterior $p(w \mid D)$ is not Gaussian and the evidence $p(D)$ is also not analytically computable. (We will deal with this later).

Application: the predictive distribution

- For a new data point $x_\bullet$, the predictive distribution for $y_\bullet$ is given by

$$\begin{aligned} p(y_\bullet = 1 \mid x_\bullet, D) &= \int p(y_\bullet = 1 \mid x_\bullet, w)\, p(w \mid D)\, \mathrm{d}w \\ &= \int \sigma(w^T x_\bullet)\, p(w \mid D)\, \mathrm{d}w \end{aligned} \tag{B-4.145}$$

- After substitution of $p(w|D)$ from B-4.142, we have closed-form expressions for both the posterior $p(w|D)$ and the predictive distribution $p(y_\bullet = 1 \mid x_\bullet, D)$. Unfortunately, these expressions contain integrals that are not analytically computable.
- Many methods have been developed to approximate the integrals in order to get analytical or numerical solutions. Here, we present the **Laplace approximation**, which is one of the simplest methods with broad applicability to Bayesian calculations.

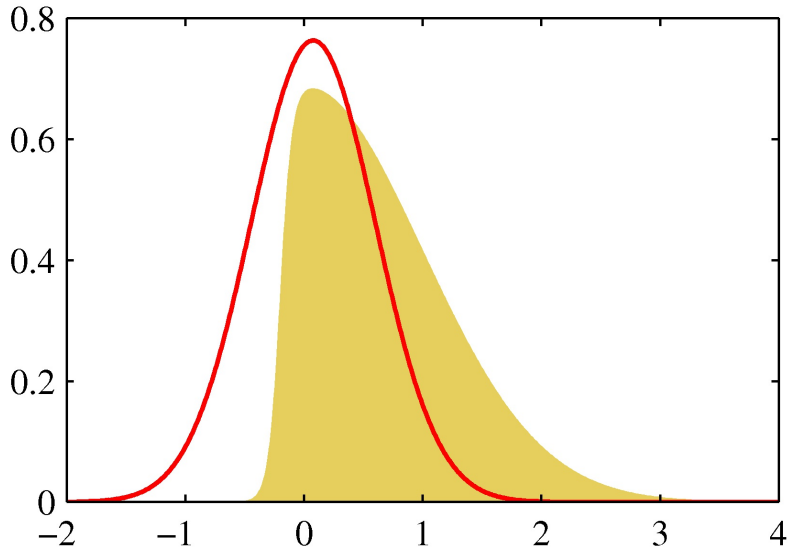
The Laplace Approximation

- The central idea of the Laplace approximation is to approximate a (possibly unnormalized) distribution $f(z)$ by a Gaussian distribution $q(z)$.
- We first give the result, followed by the derivation. The **Laplace approximation** of the distribution $f(z)$ is given by:

$$\begin{aligned} f(z) &\approx f(z_0) \exp\left(-\frac{1}{2}(z - z_0)^T A (z - z_0)\right) \\ \text{where: } z_0 &= \arg\max_z \left(\log f(z)\right) \\ A &= -\nabla\nabla \log f(z)|_{z=z_0} \end{aligned}$$

- The Laplace approximation usually serves one or both of the following two purposes:
  1. To approximate a posterior distribution without closed-form expression by a Gaussian distribution.
  2. To approximate (part of) the integrand in an integral with purpose to get an analytical solution for the integral.

Example



- (Bishop fig.4.14a). Laplace approximation (in red) to the distribution $p(z) \propto \exp(-z^2/2)\sigma(20z+4)$, where $\sigma(a) = 1/(1+e^{-a})$. The Laplace approximation is centered on the mode of $p(z)$.

Working out the Laplace Approximation

- Assume that we want to approximate a distribution $f(z)$ by a Gaussian distribution $q(z)$.
- Note that, if $q(z)$ is a Gaussian distribution, then $\log q(z)$ is a second-order polynomial in z, so we will find the Gaussian by fitting a parabola to $\log f(z)$.

estimation of mean

- The mean (z_0) of $q(z)$ is placed on the mode of $\log f(z)$, i.e.,

$$z_0 = \arg\max_z \left(\log f(z)\right) \tag{B-4.126}$$

estimation of precision matrix

- Note that since $\nabla \log f(z) = \frac{1}{f(z)} \nabla f(z)$ and the gradient $\nabla f(z)|_{z=z_0}$ vanishes at the mode $z = z_0$, we can (Taylor) expand $\log f(z)$ around $z = z_0$ as

$$\begin{aligned} \log f(z) &\approx \log f(z_0) + \overbrace{(\nabla \log f(z_0))^T (z - z_0)}^{=0 \text{ at } z=z_0} + \ldots \\ &\quad + \frac{1}{2}(z - z_0)^T \left(\nabla\nabla \log f(z_0)\right)(z - z_0) \\ &= \log f(z_0) - \frac{1}{2}(z - z_0)^T A (z - z_0) \end{aligned} \tag{B-4.131}$$

  where the Hessian matrix A is defined by

$$A = -\nabla\nabla \log f(z)|_{z=z_0} \tag{B-4.132}$$

Laplace approximation construction

- After taking exponentials in eq. B-4.131, we obtain

$$f(z) \approx f(z_0) \exp\left(-\frac{1}{2}(z - z_0)^T A (z - z_0)\right)$$

- (end derivation of Laplace Approximation)
- We can now identify $q(z)$ as

$$q(z) = \mathcal{N}\left(z \,|\, z_0, A^{-1}\right) \tag{B-4.134}$$

with z_0 and A defined by eqs. B-4.126 and B-4.132.

- All we have done up to now is approximate a function $f(z)$ by a Gaussian $q(z)$. This procedure is called the **Laplace Approximation**. Often, the required integrals (for Bayesian marginalization) can be approximately computed if we replace $f(z)$ by $q(z)$.

Bayesian Logistic Regression with the Laplace Approximation

- Let's get back to the challenge of computing the predictive class distribution (B-4.145) for Bayesian logistic regression. We first work out the Gaussian Laplace approximation $q(w)$ to the posterior weight distribution

$$\underbrace{p(w|D)}_{\text{posterior}} \propto \underbrace{\mathcal{N}(w \,|\, m_0, S_0)}_{\text{prior}} \cdot \underbrace{\prod_{n=1}^{N} \sigma\left((2y_n - 1)w^T x_n\right)}_{\text{likelihood}} \tag{B-4.142}$$

A Gausian Laplace approximation to the weights posterior

- Since we have a differentiable expression for $\log p(w|D)$, it is straightforward to compute the gradient and Hessian (for proof, see optional slide):

$$\begin{aligned} \nabla_w \log p(w|D) &= S_0^{-1} \cdot (m_0 - w) + \sum_n (2y_n - 1)(1 - \sigma_n) x_n \\ \nabla\nabla_w \log p(w|D) &= -S_0^{-1} - \sum_n \sigma_n (1 - \sigma_n) x_n x_n^T \end{aligned} \tag{B-4.143}$$

where we used shorthand σ_n for $\sigma\left((2y_n - 1)w^T x_n\right)$.

- We can now use the gradient $\nabla_w \log p(w|D)$ to find the **mode** w_N of $\log p(w|D)$ (eg by some gradient-based optimization procedure) and then use the Hessian $\nabla\nabla_w \log p(w|D)$ to get the variance of $q(w)$, leading to a **Gaussian approximate weights posterior**:

$$q(w) = \mathcal{N}\left(w \,|\, w_N, S_N\right) \tag{B-4.144}$$

with

$$S_N^{-1} = S_0^{-1} + \sum_n \sigma_n (1 - \sigma_n) x_n x_n^T \tag{B-4.143}$$

Using the Laplace-approximated parameter posterior to evaluate the predictive distribution

- In the analytically unsolveable expressions for evidence and the predictive distribution (estimating the class of a new observation), we proceed with using the Laplace approximation to the weights posterior. For a new observation $x_\bullet$, the class probability is now

$$\begin{aligned} p(y_\bullet = 1 \mid x_\bullet, D) &= \int p(y_\bullet = 1 \,|\, x_\bullet, w) \cdot p(w \,|\, D)\, \mathrm{d}w \\ &\approx \int p(y_\bullet = 1 \,|\, x_\bullet, w) \cdot \underbrace{q(w)}_{\text{Gaussian}}\, \mathrm{d}w \\ &= \int \sigma(w^T x_\bullet) \cdot \mathcal{N}\left(w \,|\, w_N, S_N\right)\, \mathrm{d}w \end{aligned} \tag{B-4.145}$$

- This looks better but we need two more clever tricks to evaluate this expression.
  1. First, note that w appears in $\sigma(w^T x_\bullet)$ as an inner product, so through substitution of $a := w^T x_\bullet$, the expression simplifies to an integral over the scalar a (see Bishop for derivation):

$$p(y_\bullet = 1 \mid x_\bullet, D) \approx \int \sigma(a)\, \mathcal{N}\left(a \mid \mu_a, \sigma_a^2\right)\, \mathrm{d}a \qquad \text{(B-4.151)}$$

$$\mu_a = w_N^T x_\bullet \qquad \text{(B-4.149)}$$

$$\sigma_a^2 = x_\bullet^T S_N x_\bullet \qquad \text{(B-4.150)}$$

  2. Secondly, while the integral of the product of a logistic function with a Gaussian is not analytically solvable, the integral of the product of a Gaussian cumulative distribution function (CDF, also known as the probit function) with a Gaussian *does* have a closed-form solution. Fortunately,

$$\Phi(\lambda a) \approx \sigma(a)$$

  with the Gaussian CDF $\Phi(x) = \frac{1}{\sqrt{(2\pi)}} \int_{-\infty}^{x} e^{-t^2/2} \mathrm{d}t$, $\lambda^2 = \pi/8$ and $\sigma(a) = 1/(1+e^{-a})$. Thus, substituting $\Phi(\lambda a)$ with $\lambda^2 = \pi/8$ for $\sigma(a)$ leads to

$$\begin{aligned} p(y_\bullet = 1 \mid x_\bullet, D) &= \int \sigma(w^T x_\bullet) \cdot p(w|D)\, \mathrm{d}w \\ &\approx \int \underbrace{\Phi(\lambda a)}_{\text{probit function}} \cdot \underbrace{\mathcal{N}\left(a \mid \mu_a, \sigma_a^2\right)}_{\text{Gaussian}}\, \mathrm{d}a \\ &= \Phi\left(\frac{\mu_a}{\sqrt{(\lambda^{-2} + \sigma_a^2)}}\right) \end{aligned} \qquad \text{(B-4.152)}$$

- We now have an approximate but **closed-form expression for the predictive class distribution for a new observation** with a Bayesian logistic regression model.

- Note that, by Eq.B-4.143, the variance S_N (and consequently σ_a^2) for the weight vector depends on the distribution of the training set. Large uncertainty about the weights (in areas with little training data and uninformative prior variance S_0) increases σ_a^2 and takes the posterior class probability eq. B-4.152 closer to 0.5. Does that make sense?

- Apparently, the Laplace approximation leads to a closed-form solutions for Bayesian logistic regression (although admittedly, the derivation is no walk in the park).

- Exam guide: The derivation of closed-form expression eq. B-4.152 for the predictive class distribution requires clever tricks and is therefore not something that you should be able to reproduce at the exam without assistance. You should understand the Laplace Approximation though and be able to work out simpler examples.

ML Estimation for Discriminative Classification

- Rather than the computationally involved Laplace approximation for Bayesian inference, in practice, discriminative classification is often executed through maximum likelihood estimation.

- With the usual 1-of-K encoding scheme for classes ($y_{nk} = 1$ if $x_n \in \mathcal{C}_k$, otherwise $y_{nk} = 0$), the log-likelihood for a K-dimensional discriminative classifier is

$$\begin{aligned} \mathrm{L}(\theta) &= \log \prod_n \prod_k p(\mathcal{C}_k|x_n, \theta)^{y_{nk}} \\ &= \log \prod_n \prod_k \left(\underbrace{\frac{e^{\theta_k^T x_n}}{\sum_j e^{\theta_j^T x_n}}}_{\text{softmax function}} \right)^{y_{nk}} \\ &= \sum_n \sum_k y_{kn} \log \left(\frac{e^{\theta_k^T x_n}}{\sum_j e^{\theta_j^T x_n}}\right) \end{aligned}$$

- Computing the gradient $\nabla_{\theta_k} \mathrm{L}(\theta)$ leads to (for proof, see optional slide below)

$$\nabla_{\theta_k} \mathrm{L}(\theta) = \sum_n \underbrace{\Big(\underbrace{y_{nk}}_{\text{target}} - \underbrace{\frac{e^{\theta_k^T x_n}}{\sum_j e^{\theta_j^T x_n}}}_{\text{prediction}} \Big)}_{\text{prediction error}} \cdot x_n$$

- Compare this to the gradient for *linear* regression:

$$\nabla_\theta \mathrm{L}(\theta) = \sum_n \left(y_n - \theta^T x_n \right) x_n$$

- In both cases

$$\nabla_\theta \mathrm{L} = \sum_n (\text{target}_n - \text{prediction}_n) \cdot \text{input}_n$$

- The parameter vector θ for logistic regression can be estimated through iterative gradient-based adaptation. E.g. (with iteration index i),

$$\hat{\theta}^{(i+1)} = \hat{\theta}^{(i)} + \eta \cdot \nabla_\theta \mathrm{L}(\theta)\big|_{\theta=\hat{\theta}^{(i)}}$$

  - Note that, while in the Bayesian approach we get to update θ with **Kalman-gain-weighted** prediction errors (which is optimal), in the maximum likelihood approach, we weigh the prediction errors with **input** values (which is less precise).

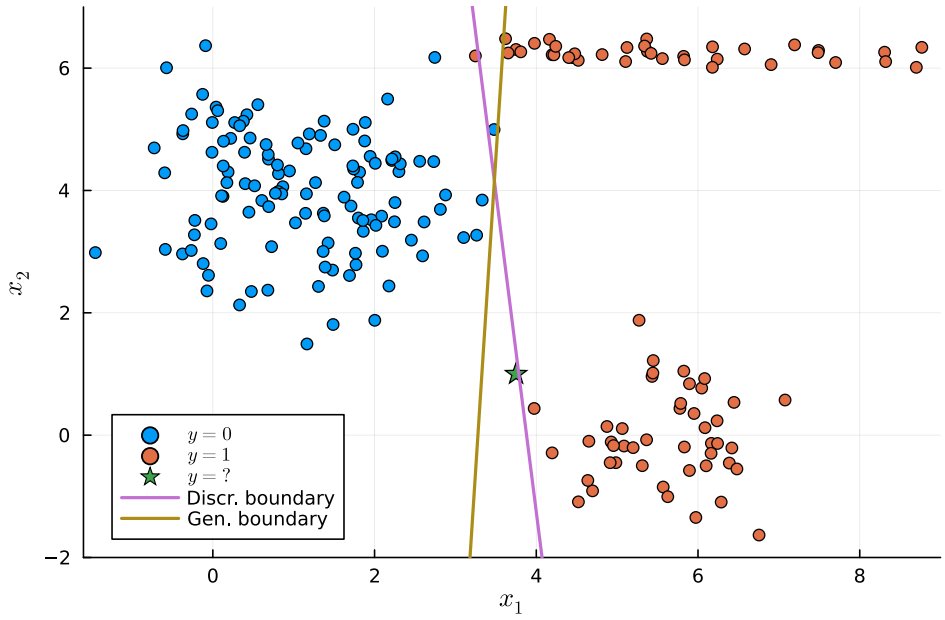
Code Example: ML Estimation for Discriminative Classification

- Let us perform ML estimation of w on the data set from the introduction. To allow an offset in the discrimination boundary, we add a constant 1 to the feature vector x. We only have to specify the (negative) log-likelihood and the gradient w.r.t. w. Then, we use an off-the-shelf optimisation library to minimize the negative log-likelihood.
- We plot the resulting maximum likelihood discrimination boundary. For comparison we also plot the ML discrimination boundary obtained from the code example in the generative Gaussian classifier lesson.

In [3]:
```
using Optim # Optimization library

    y_1 = zeros(length(y))# class 1 indicator vector
    y_1[findall(y)] .= 1
    X_ext = vcat(X, ones(1, length(y))) # Extend X with a row of ones to allow an offset in the discrimination b

    # Implement negative log-likelihood function
    function negative_log_likelihood(θ::Vector)
        # Return negative log-likelihood: -L(θ)
        p_1 = 1.0 ./ (1.0 .+ exp.(-X_ext' * θ))    # P(C1|X,θ)
        return -sum(log.( (y_1 .* p_1) + ((1 .- y_1).*(1 .- p_1))) ) # negative log-likelihood
    end

    # Use Optim.jl optimiser to minimize the negative log-likelihood function w.r.t. θ
    results = optimize(negative_log_likelihood, zeros(3), LBFGS())
    θ = results.minimizer

    # Plot the data set and ML discrimination boundary
    plotDataSet()
    p_1(x) = 1.0 ./ (1.0 .+ exp(-([x;1.]' * θ)))
    boundary(x1) = -1 ./ θ[2] * (θ[1]*x1 .+ θ[3])

    generative_boundary = buildGenerativeDiscriminationBoundary(X, y)


    x_test = [3.75;1.0]
    println("P(C1|x•,θ) = $(p_1(x_test))")
    plot!([-2., 10.], boundary.([-2., 10.]), label="Discr. boundary", linewidth=2)
    plot!([-2.,10.], generative_boundary.([-2,10]), label="Gen. boundary", linewidth=2)
```

```
P(C1|x•,θ) = 0.4016963543060162
```

Out[3]:



- The generative model gives a bad result because the feature distribution of one class is clearly non-Gaussian: the model does not fit the data well.
- The discriminative approach does not suffer from this problem because it makes no assumptions about the feature distribution $p(x)$. Rather, it just estimates the conditional class distribution $p(y|x)$ directly.

Why be Bayesian?

- Why should you embrace the Bayesian approach to logistic regression? After all, Maximum Likelihood for logistic regression seems simpler.
- Still, consider the following:
  - Bayesian logistic regression with the Laplace approximation ultimately leads to very simple analytic rules. Moreover, modern probabilistic programming languages and packages are able to automate the above inference derivations. (We just do them here to gain insight in difficult inference processes.)
  - Bayesian logistic regression offers the option to compute model evidence.
  - Bayesian logistic regression processes uncertainties, e.g., in places where almost no data is observed, the posterior class probability will pull back to the prior class probability rather than predicting some arbitrary probability.

Recap Classification

	Generative	**Discriminative (ML)**
1	Like **density estimation**, model joint prob. $p(\mathcal{C}_k)p(x\|\mathcal{C}_k) = \pi_k \mathcal{N}(\mu_k, \Sigma)$	Like (linear) **regression**, model conditional $p(\mathcal{C}_k\|x, \theta)$
2	Leads to **softmax** posterior class probability $p(\mathcal{C}_k\|x, \theta) = e^{\theta_k^T x}/Z$ with **structured** θ	**Choose** also softmax posterior class probability $p(\mathcal{C}_k\|x, \theta) = e^{\theta_k^T x}/Z$ but now with 'free' θ
3	For Gaussian $p(x\|\mathcal{C}_k)$ and multinomial priors, $\hat{\theta}_k = \begin{bmatrix} -\frac{1}{2}\mu_k^T \sigma^{-1} \mu_k + \log \pi_k \\ \sigma^{-1}\mu_k \end{bmatrix}$ **in one shot**.	Find $\hat{\theta}_k$ through gradient-based adaptation $\nabla_{\theta_k} \mathrm{L}(\theta) = \sum_n \left(y_{nk} - \frac{e^{\theta_k^T x_n}}{\sum_{k'} e^{\theta_{k'}^T x_n}} \right) x_n$

OPTIONAL SLIDES

Proof of gradient and Hessian for Laplace Approximation of Posterior

- We will start with the posterior

$$\underbrace{p(w|D)}_{\text{posterior}} \propto \underbrace{\mathcal{N}(w\,|\,m_0, S_0)}_{\text{prior}} \cdot \underbrace{\prod_{n=1}^{N} \sigma\big(\underbrace{(2y_n-1)w^T x_n}_{a_n}\big)}_{\text{likelihood}} \tag{B-4.142}$$

  from which it follows that

$$\log p(w|D) \propto -\frac{1}{2}\log|S_0| - \frac{1}{2}(w-m_0)^T S_0^{-1}(w-m_0) + \sum_n \log \sigma(a_n)$$

  and the gradient

$$\begin{aligned}\nabla_w \log p(w|D) &\propto \underbrace{S_0^{-1}(m_0-w)}_{\text{SRM-5b}} + \sum_n \underbrace{\frac{1}{\sigma(a_n)}}_{\frac{\partial \log\sigma(a_n)}{\partial\sigma(a_n)}} \cdot \underbrace{\sigma(a_n)\cdot(1-\sigma(a_n))}_{\frac{\partial\sigma(a_n)}{\partial a_n}} \cdot \underbrace{(2y_n-1)x_n}_{\frac{\partial a_n}{\partial w}\ \text{(see SRM-5a)}} \\ &= S_0^{-1}(m_0-w) + \sum_n (2y_n-1)(1-\sigma(a_n))x_n \quad \text{(gradient)}\end{aligned}$$

  where we used $\sigma'(a) = \sigma(a)\cdot(1-\sigma(a))$.

- For the Hessian, we continue to differentiate the transpose of the gradient, leading to

$$\begin{aligned}\nabla\nabla_w \log p(w|D) &= \nabla_w\big(S_0^{-1}(m_0-w)\big)^T - \sum_n (2y_n-1)x_n \nabla_w \sigma(a_n)^T \\ &= -S_0^{-1} - \sum_n (2y_n-1)x_n \cdot \underbrace{\sigma(a_n)\cdot(1-\sigma(a_n))}_{\frac{\partial\sigma(a_n)^T}{\partial a_n^T}} \cdot \underbrace{(2y_n-1)x_n^T}_{\frac{\partial a_n^T}{\partial w}} \\ &= -S_0^{-1} - \sum_n \sigma(a_n)\cdot(1-\sigma(a_n))\cdot x_n x_n^T \quad \text{(Hessian)}\end{aligned}$$

  since $(2y_n-1)^2 = 1$ for $y_n \in \{0,1\}$.

Proof of Derivative of Log-likelihood for Logistic Regression

- The Log-likelihood is $\mathrm{L}(\theta) = \log \prod_n \prod_k \underbrace{p(\mathcal{C}_k|x_n,\theta)}_{p_{nk}}^{y_{nk}} =$

  $\sum_{n,k} y_{nk} \log p_{nk}$

- Use the fact that the softmax $\phi_k \equiv e^{a_k}/\sum_j e^{a_j}$ has analytical derivative:

$$\begin{aligned}\frac{\partial \phi_k}{\partial a_j} &= \frac{(\sum_j e^{a_j})e^{a_k}\delta_{kj} - e^{a_j}e^{a_k}}{(\sum_j e^{a_j})^2} = \frac{e^{a_k}}{\sum_j e^{a_j}}\delta_{kj} \\ &\quad - \frac{e^{a_j}}{\sum_j e^{a_j}}\frac{e^{a_k}}{\sum_j e^{a_j}} \\ &= \phi_k \cdot (\delta_{kj} - \phi_j)\end{aligned}$$

- Take the derivative of $\mathrm{L}(\theta)$ (or: how to spend a hour ...)

$$\begin{aligned}\nabla_{\theta_j}\mathrm{L}(\theta) &= \sum_{n,k} \frac{\partial \mathrm{L}_{nk}}{\partial p_{nk}} \cdot \frac{\partial p_{nk}}{\partial a_{nj}} \cdot \frac{\partial a_{nj}}{\partial \theta_j} \\ &= \sum_{n,k} \frac{y_{nk}}{p_{nk}} \cdot p_{nk}(\delta_{kj} - p_{nj}) \cdot x_n \\ &= \sum_n \Big(y_{nj}(1 - p_{nj}) - \sum_{k \neq j} y_{nk} p_{nj} \Big) \cdot x_n \\ &= \sum_n \big(y_{nj} - p_{nj} \big) \cdot x_n \\ &= \sum_n \Big(\underbrace{y_{nj}}_{\text{target}} - \underbrace{\frac{e^{\theta_j^T x_n}}{\sum_{j'} e^{\theta_{j'}^T x_n}}}_{\text{prediction}} \Big) \cdot x_n\end{aligned}$$

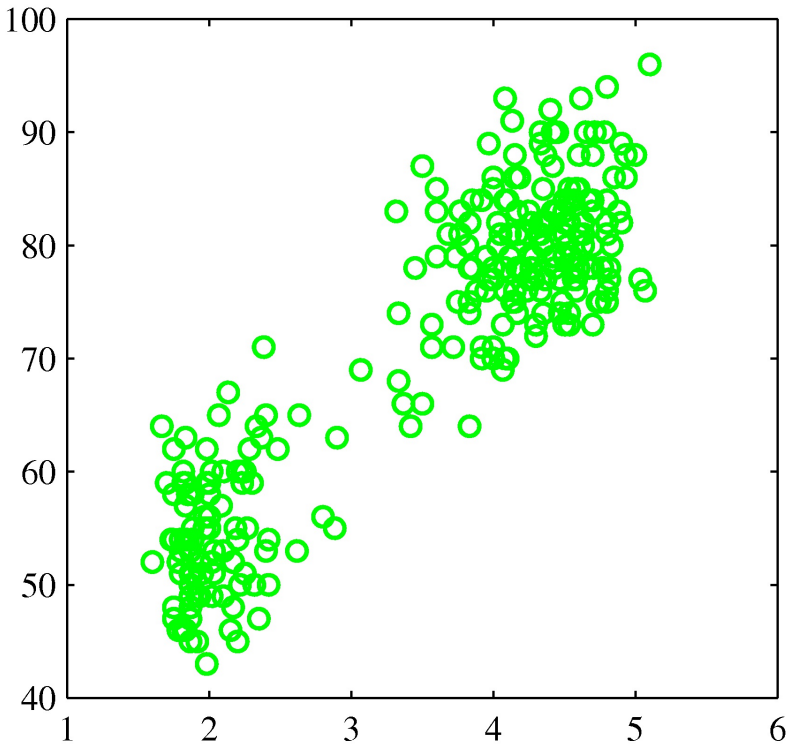
Latent Variable Models and Variational Bayes

Preliminaries

- Goal
  - Introduction to latent variable models and variational inference by Free energy minimization
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop (2016), pp. 461-486 (sections 10.1, 10.2 and 10.3)
    - Ariel Caticha (2010), Entropic Inference
      - tutorial on entropic inference, which is a generalization to Bayes rule and provides a foundation for variational inference.
  - references
    - Blei et al. (2017), Variational Inference: A Review for Statisticians
    - Lanczos (1961), The variational principles of mechanics
    - Senoz et al. (2021), Variational Message Passing and Local Constraint Manipulation in Factor Graphs
    - Dauwels (2007), On variational message passing on factor graphs
    - Shore and Johnson (1980), Axiomatic Derivation of the Principle of Maximum Entropy and the Principle of Minimum Cross-Entropy

Challenge : Density Modeling for the Old Faithful Data Set

- You're now asked to build a density model for a data set (Old Faithful, Bishop pg. 681) that clearly is not distributed as a single Gaussian:



Unobserved Classes

- Consider again a set of observed data $D = \{x_1, \ldots, x_N\}$
- This time we suspect that there are *unobserved* class labels that would help explain (or predict) the data, e.g.,
  - the observed data are the color of living things; the unobserved classes are animals and plants.
  - observed are wheel sizes; unobserved categories are trucks and personal cars.
  - observed is an audio signal; unobserved classes include speech, music, traffic noise, etc.
- Classification problems with unobserved classes are called **Clustering** problems. The learning algorithm needs to **discover the classes from the observed data**.

The Gaussian Mixture Model

- The spread of the data in the illustrative example looks like it could be modeled by two Gaussians. Let's develop a model for this data set.
- Let $D = \{x_n\}$ be a set of observations. We associate a one-hot coded hidden class label z_n with each observation:

$$z_{nk} = \begin{cases} 1 & \text{if } x_n \in \mathcal{C}_k \text{ (the } k\text{-th class)} \\ 0 & \text{otherwise} \end{cases}$$

- We consider the same model as we did in the generative classification lesson: the data for each class is distributed as a Gaussian:

$$\begin{aligned} p(x_n|z_{nk} = 1) &= \mathcal{N}\left(x_n|\mu_k, \Sigma_k\right) \\ p(z_{nk} = 1) &= \pi_k \end{aligned}$$

  which can be summarized with the selection variables z_{nk} as

$$p(x_n, z_n) = \prod_{k=1}^{K} \underbrace{\left(\pi_k \cdot \mathcal{N}\left(x_n|\mu_k, \Sigma_k\right)\right)}_{p(x_n, z_{nk}=1)}^{z_{nk}}$$

- *Again*, this is the same model as we defined for the generative classification model: A Gaussian-Categorical model but now with unobserved classes.
- This model (with **unobserved class labels**) is known as a **Gaussian Mixture Model** (GMM).

The Marginal Distribution for the GMM

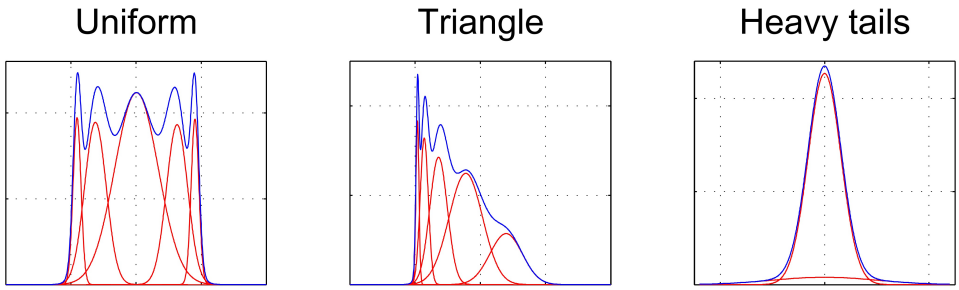
- In the literature, the GMM is often introduced by the marginal distribution for an *observed* data point x_n, given by

$$\begin{aligned} p(x_n) &= \sum_{z_n} p(x_n, z_n) \\ &= \sum_{k=1}^{K} \pi_k \cdot \mathcal{N}\left(x_n | \mu_k, \Sigma_k\right) \end{aligned} \tag{B-9.12}$$

- Full proof as an exercise.
- Eq. B-9.12 reveals the link to the name Gaussian *mixture model*. The priors π_k for the k-th class are also called **mixture coefficients**.
- Be aware that Eq. B-9.12 is not the generative model for the GMM! The generative model is the joint distribution $p(x, z)$ over all variables, including the latent variables.

GMM is a very flexible model

- GMMs are very popular models. They have decent computational properties and are **universal approximators of densities** (as long as there are enough Gaussians of course)



- (In the above figure, the Gaussian components are shown in red and the pdf of the mixture models in blue).

Latent Variable Models

- The GMM contains both *observed* variables $\{x_n\}$, (unobserved) *parameters* $\theta = \{\pi_k, \mu_k, \Sigma_k\}$ *and* unobserved (synonym: latent, hidden) variables $\{z_{nk}\}$.
- From a Bayesian viewpoint, both latent variables $\{z_{nk}\}$ and parameters θ are just unobserved variables for which we can set a prior and compute a posterior by Bayes rule.
- Note that z_{nk} has a subscript n, hence its value depends not only on the class (k) but also on the n-th observation (in contrast to parameters). These observation-dependent latent variables are generally a useful tool to encode additional structure in the model about the causes of your observations. Here (in the GMM), the latent variables $\{z_{nk}\}$ encode (unobserved) class membership.
- Models with observation-dependent latent variables are generally called **Latent Variable Models**.
- By adding model structure through (equations among) observation-dependent latent variables, we can often build more accurate models for very complex processes. Unfortunately, adding structure through observation-dependent latent variables in models often is accompanied by a more complex inference task.

Inference for GMM is Difficult

- Indeed, the fact that the observation-dependent class labels are *unobserved* for the GMM, leads to a problem for processing new data by Bayes rule in a GMM.
- Consider a given data set $D = \{x_n\}$. We recall here the log-likelihood for the Gaussian-Categorial Model, see the generative classification lesson:

$$\log\, p(D|\theta) = \sum_{n,k} y_{nk} \underbrace{\log \mathcal{N}(x_n|\mu_k, \Sigma)}_{\text{Gaussian}} + \underbrace{\sum_{n,k} y_{nk} \log \pi_k}_{\text{multinomial}} .$$

- Since the class labels $y_{nk} \in \{0, 1\}$ were assumed to be given by the data set, maximization of this expression decomposed into a set of simple update rules for the Gaussian and multinomial distributions.
- However, for the Gaussian mixture model (same log-likelihood function with z_{nk} replacing y_{nk}), the class labels $\{z_{nk}\}$ are *unobserved* and they need to be estimated alongside with the parameters.
- There is no known conjugate prior for the latent variables for the GMM likelihood function and, therefore, we cannot compute Bayes rule to get a closed-form expression for the posterior over the latent variables:

$$\begin{aligned} &\underbrace{p(\{z_{nk}\}, \{\mu_k, \Sigma_k, \pi_k\}|D)}_{\text{posterior (no analytical solution)}} \\ &\propto \underbrace{p(D \,|\, \{z_{nk}\}, \{\mu_k, \Sigma_k, \pi_k\})}_{\text{likelihood}} \\ &\cdot \underbrace{p(\{z_{nk}\}, \{\mu_k, \Sigma_k, \pi_k\})}_{\text{prior (no known conjugate)}} \end{aligned}$$

- Can we still compute an approximate posterior? In this lesson, we introduce an approximate Bayesian inference method known as **Variational Bayes** (VB) (also known as **Variational Inference**) that can be used for Bayesian inference in models with latent variables. Later in this lesson, we will use VB to do inference in the GMM.

The Variational Free Energy Functional

- We'll start from scratch. Consider a model $p(x, z) = p(x|z)p(z)$, where x and z are observed and latent variables respectively. z may include parameters but also observation-dependent latent variables.
- The goal of Bayesian inference is to transform the (known) *likelihood-times-prior* factorization of the full model to a *posterior-times-evidence* decomposition:

$$\underbrace{p(x|z)p(z)}_{\text{what we know}} \rightarrow \underbrace{p(z|x)p(x)}_{\text{what we want}}$$

- Remember from the Bayesian machine learning lesson that negative log-evidence can be decomposed as "complexity" minus "accuracy" terms (the CA decomposition):

$$\begin{aligned} -\log p(x) = &\underbrace{\int p(z|x) \log \frac{p(z|x)}{p(z)} \mathrm{d}z}_{\text{complexity}} \\ &- \underbrace{\int p(z|x) \log p(x|z) \mathrm{d}z}_{\text{accuracy}} \end{aligned}$$

- The CA decomposition cannot be evaluated because it depends on the posterior $p(z|x)$, which cannot be evaluated since it is the objective of the inference process.
- Let's now introduce a distribution $q(z)$ that we use to approximate the posterior $p(z|x)$, and assume that $q(z)$ can be evaluated!
- If will substitute $q(z)$ for $p(z|x)$ in the CA decomposition, then we obtain

$$F[q] \triangleq \underbrace{\int q(z) \log \frac{q(z)}{p(z)} \mathrm{d}z}_{\text{complexity}} \\ \underbrace{- \int q(z) \log p(x|z) \mathrm{d}z}_{\text{accuracy}}$$

- This expression is called the variational *Free Energy* (FE). We consider the Free Energy F as a function of the posterior $q(z)$. Technically, a function of a function is called a functional, and we write square brackets (e.g., $F[q]$) to differentiate functionals from functions (e.g., $q(z)$).
- Note that all factors in the CA decomposition of FE (i.e., $q(z)$, $p(z)$ and $p(x|z)$) can be evaluated as a function of z (and x is observed), and therefore the FE can be evaluated. This is important: log-evidence cannot be evaluated, but FE *can* be evaluated!

Inference by FE Minimization

- It turns out that we can do (approximate) Bayesian inference through FE Minimization (FEM) with respect to q.
- To explain inference by FEM, we first rewrite FE in terms of "inference bound" minus "log-evidence" terms (the bound-evidence (BE) decomposition):

$$\begin{aligned} F[q] &= \underbrace{\int q(z) \log \frac{q(z)}{p(z)} \mathrm{d}z}_{\text{complexity}} - \underbrace{\int q(z) \log p(x|z) \mathrm{d}z}_{\text{accuracy}} \\ &= \int q(z) \log \frac{q(z)}{p(z)p(x|z)} \mathrm{d}z \\ &= \int q(z) \log \frac{q(z)}{p(z|x)p(x)} \mathrm{d}z \\ &= \underbrace{\int q(z) \log \frac{q(z)}{p(z|x)} \mathrm{d}z}_{\text{inference bound} \geq 0} - \underbrace{\log p(x)}_{\text{log-evidence}} \end{aligned}$$

- Note that the inference bound is a Kullback-Leibler (KL) divergence between an (approximate) posterior $q(z)$ and the (perfect) Bayesian posterior $p(z|x)$. See this slide in the BML Class for more info on the KL divergence.
- Since the second term (log-evidence) does not involve $q(z)$, FEM over q will bring $q(z)$ closer to the Bayesian posterior $p(z|x)$.
- Since $\mathrm{KL}[q(z), p(z|x)] \geq 0$ for any $q(z)$, and $\mathrm{KL}[q(z), p(z|x)] = 0$ only if $q(z) = p(z|x)$, the FE is always an upperbound on (minus) log-evidence, i.e.,

$$F[q] \geq -\log p(x)\,.$$

- As a result, **global FEM recovers Bayes rule**, i.e., global optimization of FE w.r.t. q leads to

$$q^*(z) = \arg\min_q F[q]$$

where

$$\begin{aligned} &\text{posterior: } q^*(z) = p(z|x) \\ &\text{evidence: } F[q^*] = -\log p(x) \end{aligned}$$

- In practice, even if we cannot attain the global minimum of FE, we can still use a local minimum

$$\hat{q}(z) \approx \arg\min_q F[q]$$

to accomplish **approximate Bayesian inference** by:

$$\begin{aligned} &\text{posterior: } \hat{q}(z) \approx p(z|x) \\ &\text{evidence: } F[\hat{q}] \approx -\log p(x) \end{aligned}$$

- In short, FE minimization transforms an inference problem (that involves integration) to an optimization problem! Generally, optimization problems are easier to solve than integration problems.

- Executing inference by minimizing the variational FE functional is called **Variational Bayes** (VB) or variational inference.
- (As an aside), note that Bishop introduces in Eq. B-10.3 an *Evidence Lower BOund* (in modern machine learning literature abbreviated as **ELBO**) $\mathcal{L}[q]$ that equals the *negative* FE ($\mathcal{L}[q] = -F[q]$). In this class, we prefer to discuss inference in terms of minimizing Free Energy rather than maximizing ELBO, but note that these two concepts are equivalent. (The reason why we prefer the Free Energy formulation relates to the terminology in the Free Energy Principle, which we introduce in the Intelligent Agents and active Inference lesson (B12)).

Constrained FE Minimization

- It is common to add simplifying constraints to optimization problems to make a difficult optimization task tractible. This is also common practice when approximating Bayesian inference by FE minimization.
- There are three important cases of adding constraints to $q(z)$ that often alleviates the FE minimization task:

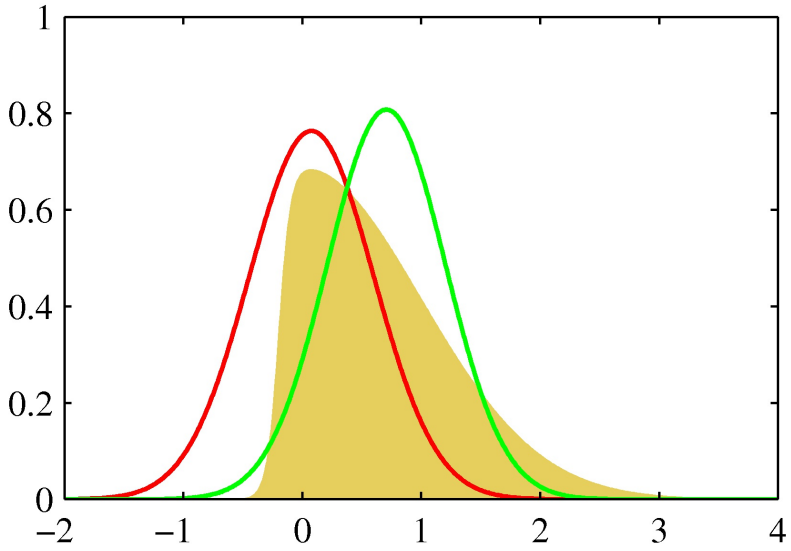
1. #### form constraints
   - For almost every practical setting, we constrain the posterior $q(z)$ to be a specific parameterized probability distribution, e.g.,

$$q(z) = \mathcal{N}\left(z|\mu, \Sigma\right) \,.$$

   - In this case, the *functional* minimization problem for $F[q]$ reduces to the minimization of a *function*

$$F(\mu, \Sigma) = \int \mathcal{N}\left(z|\mu, \Sigma\right) \log \frac{\mathcal{N}\left(z|\mu, \Sigma\right)}{p(x, z)} \mathrm{d}z$$

     w.r.t. the parameters μ and Σ.
   - We can often use standard gradient-based optimization methods to minimize the FE.
   - In the figure below (see Bishop Fig.10.1a, pg.464), an intractable Bayesian posterior (yellow) for a binary classification problem has been approximated by a Laplace approximation (red) and a variational posterior $q(z) \sim \mathcal{N}(\mu, \sigma^2)$ (green).



2. #### factorization constraints
   - In addition to form constraints, it is also common to constrain the posterior $q(z)$ by a specific factorization. For instance, in *mean-field factorization*, we constrain the posterior to factorize into a set of independent factors, i.e.,

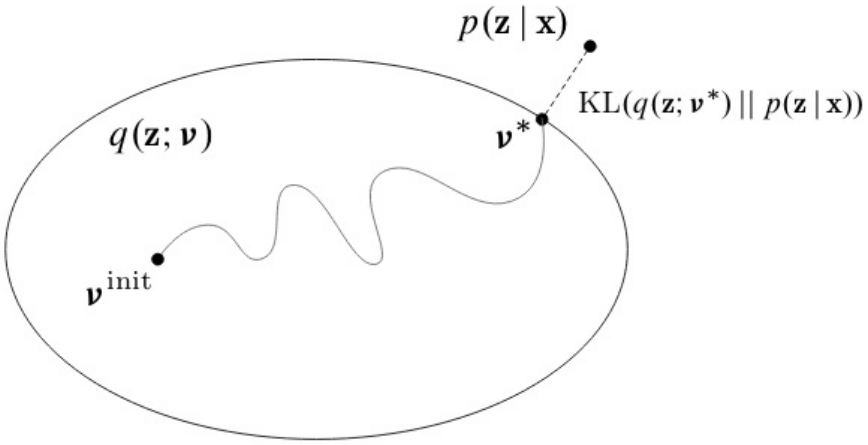
$$q(z) = \prod_{j=1}^{m} q_j(z_j) \,, \tag{B-10.5}$$

   - Variational inference with mean-field factorization has been worked out in detail as the **Coordinate Ascent Variational Inference** (CAVI) algorithm. See the Optional Slide on CAVI for details.
   - Mean-field factorization is just an example of various *factorization constraints* that have been successfully applied to FEM.
3. #### other constraints, e.g., the Expectation-Minimization (EM) algorithm

- Aside from form and factorization constraints, several ad hoc algorithms have been derived that ease the process of FE minimization for particular models.
- In particular, the Expectation-Maximization (EM) algorithm is a famous special case of constrained FE minimization. The EM algorithm places some constraints on both the posterior $q(z)$ and the prior $p(z)$ (see the OPTIONAL SLIDE for more info) that essentially reduces FE minimization to maximum likelihood estimation.

Visualization of Constrained Free Energy Minimization

- The following image by David Blei illustrates the Variational Bayes approach:



-

- The Bayesian posterior $p(z|x)$ (upper-right) is the posterior that would be obtained through executing Bayes rule, but unfortunately Bayes rule is not tractable here. Instead, we propose a variational posterior $q(z;\nu)$ that is parameterized by ν. The inside area of the ellipsis represents the area that is reachable by choosing values for the parameter ν. Note that $p(z|x)$ is not reachable. We start the FE minimization process by choosing an initial value ν^{init}, which corresponds to posterior $q(z;\nu^{\text{init}})$, as indicated in the figure. FE minimization leads to a final value ν^* that minimizes the KL-divergence between $q(z;\nu)$ and $p(z|x)$.

Challenge Revisited: Density Modeling for the Old Faithful Data Set

- Let's get back to the illustrative challenge at the beginning of this lesson: we want to do density modeling for the Old Faithful data set.

model specification

- We consider a Gaussian Mixture Model, specified by

$$\begin{aligned} p(x,z|\theta) &= p(x|z,\mu,\Lambda)p(z|\pi) \\ &= \prod_{n=1}^{N}\prod_{k=1}^{K}\mathcal{N}\left(x_n|\mu_k,\Lambda_k^{-1}\right)^{z_{nk}} \cdot \prod_{n=1}^{N}\prod_{k=1}^{K}\pi_k^{z_{nk}} \\ &= \prod_{n=1}^{N}\prod_{k=1}^{K}\left(\pi_k \cdot \mathcal{N}\left(x_n|\mu_k,\Lambda_k^{-1}\right)\right)^{z_{nk}} \end{aligned} \qquad \text{(B-10.37,38)}$$

- Let us introduce some priors for the parameters π, μ and Λ. We factorize the prior and choose conjugate distributions by

$$p(\pi,\mu,\Lambda) = p(\pi)p(\mu|\Lambda)p(\Lambda)$$

with

$$p(\pi) = \mathrm{Dir}(\pi|\alpha_0) = C(\alpha_0)\prod_k \pi_k^{\alpha_0-1} \qquad \text{(B-10.39)}$$

$$p(\mu|\Lambda) = \prod_k \mathcal{N}\left(\mu_k|m_0,(\beta_0\Lambda_k)^{-1}\right) \qquad \text{(B-10.40)}$$

$$p(\Lambda) = \prod_k \mathcal{W}(\Lambda_k|W_0,\nu_0) \qquad \text{(B-10.40)}$$

where $\mathcal{W}(\cdot)$ is a Wishart distribution (i.e., a multi-dimensional Gamma distribution).

- The full generative model is now specified by

$$\begin{aligned} &p(x, z, \pi, \mu, \Lambda) \\ &= p(x|z, \mu, \Lambda) p(z|\pi) p(\pi) p(\mu|\Lambda) p(\Lambda) \end{aligned} \tag{B-10.41}$$

with hyperparameters $\{\alpha_0, m_0, \beta_0, W_0, \nu_0\}$.

inference

- Assume that we have observed $D = \{x_1, x_2, \ldots, x_N\}$ and are interested to infer a posterior distribution for the parameters π, μ and Λ.
- We will approximate Bayesian inference by FE minimization. For the specified model, this leads to FE minimization w.r.t. the hyperparameters, i.e., we need to minimize the function

$$F(\alpha_0, m_0, \beta_0, W_0, \nu_0)\,.$$

- In general, this function can be optimized in various ways, e.g. by a gradient-descent procedure.

- It turns out that adding the following **factorization constraints** on the posterior makes the FEM task analytically tractible:

$$q(z, \pi, \mu, \Lambda) = q(z) \cdot q(\pi, \mu, \Lambda)\,. \tag{B-10.42}$$

- For this specific case (GMM model with assumed factorization and parameterization constraints), Bishop shows that the equations for the optimal solutions (Eq. B-10.9) are analytically solvable, leading to the following variational update equations (for $k = 1, \ldots, K$):

$$\alpha_k = \alpha_0 + N_k \tag{B-10.58}$$

$$\beta_k = \beta_0 + N_k \tag{B-10.60}$$

$$m_k = \frac{1}{\beta_k}(\beta_0 m_0 + N_k \bar{x}_k) \tag{B-10.61}$$

$$\begin{aligned} W_k^{-1} &= W_0^{-1} + N_k S_k \\ &\quad + \frac{\beta_0 N_k}{\beta_0 + N_k}(\bar{x}_k - m_0)\,(\bar{x}_k - m_0)^T \end{aligned} \tag{B-10.62}$$

$$\nu_k = \nu_0 + N_k \tag{B-10.63}$$

where we used

$$\begin{aligned} \log \rho_{nk} &= \mathbb{E}\left[\log \pi_k\right] + \frac{1}{2}\mathbb{E}\left[\log |\Lambda_k|\right] - \frac{D}{2}\log(2\pi) \\ &\quad - \frac{1}{2}\mathbb{E}\left[(x_k - \mu_k)^T \Lambda_k (x_k - \mu_k)\right] \end{aligned} \tag{B-10.46}$$

$$r_{nk} = \frac{\rho_{nk}}{\sum_{j=1}^K \rho_{nj}} \tag{B-10.49}$$

$$N_k = \sum_{n=1}^N r_{nk} x_n \tag{B-10.51}$$

$$\bar{x}_k = \frac{1}{N_k} \sum_{n=1}^N r_{nk} x_n \tag{B-10.52}$$

$$S_k = \frac{1}{N_k} \sum_{n=1}^N r_{nk}\,(x_n - \bar{x}_k)\,(x_n - \bar{x}_k)^T \tag{B-10.53}$$

- Exam guide: Working out FE minimization for the GMM to these update equations (eqs B-10.58 through B-10.63) is not something that you need to reproduce without assistance at the exam. Rather, the essence is that *it is possible* to arrive at closed-form variational update equations for the GMM. You should understand though how FEM works conceptually and in principle be able to derive variational update equations for very simple models that do not involve clever mathematical tricks.

Code Example: FEM for GMM on Old Faithfull data set

- Below we exemplify training of a Gaussian Mixture Model on the Old Faithful data set by Free Energy Minimization, using the constraints as specified above.

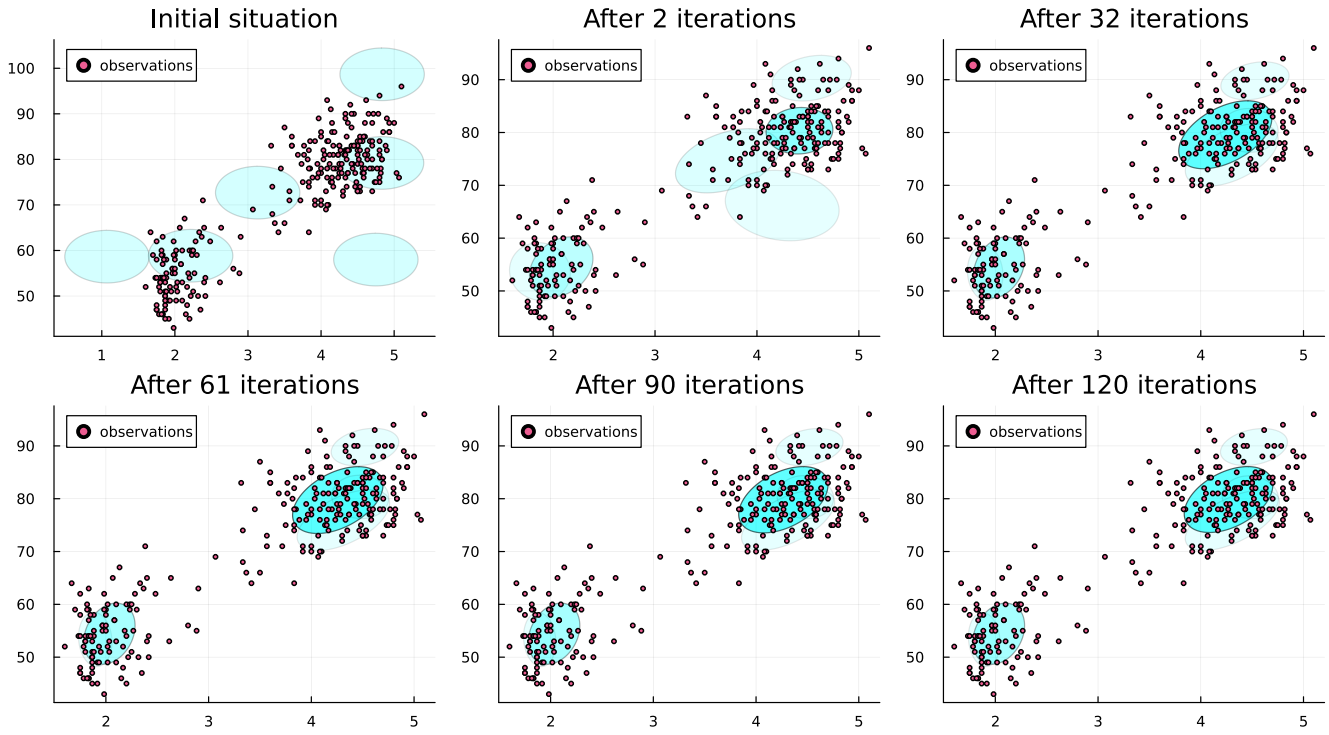
In [1]:
```
using Pkg; Pkg.activate("../."); Pkg.instantiate();
using IJulia; try IJulia.clear_output(); catch _ end
```

Out[1]: 0

In [2]:
```
using DataFrames, CSV, LinearAlgebra, PDMats, SpecialFunctions, Random

include("scripts/gmm_plot.jl") # Holds plotting function
old_faithful = CSV.read("datasets/old_faithful.csv",DataFrame);
X = convert(Matrix{Float64}, [old_faithful[!,1] old_faithful[!,2]]');#data matrix
N = size(X, 2) #number of observations
K = 6

function sufficientStatistics(X,r,k::Int) #function to compute sufficient statistics
    N_k = sum(r[k,:])
    hat_x_k = sum([r[k,n]*X[:,n] for n in 1:N]) ./ N_k
    S_k = sum([r[k,n]*(X[:,n]-hat_x_k)*(X[:,n]-hat_x_k)' for n in 1:N]) ./ N_k
    return N_k, hat_x_k, S_k
end

function updateMeanPrecisionPi(m_0,β_0,W_0,ν_0,α_0,r) #variational maximisation function
    m = Array{Float64}(undef,2,K) #mean of the clusters
    β = Array{Float64}(undef,K) #precision scaling for Gausian distribution
    W = Array{Float64}(undef,2,2,K) #precision prior for Wishart distributions
    ν = Array{Float64}(undef,K) #degrees of freedom parameter for Wishart distribution
    α = Array{Float64}(undef,K) #Dirichlet distribution parameter
    for k=1:K
        sst = sufficientStatistics(X,r,k)
        α[k] = α_0[k] + sst[1]; β[k] = β_0[k] + sst[1]; ν[k] = ν_0[k] .+ sst[1]
        m[:,k] = (1/β[k])*(β_0[k].*m_0[:,k] + sst[1].*sst[2])
        W[:,:,k] = inv(inv(W_0[:,:,k])+sst[3]*sst[1] + ((β_0[k]*sst[1])/(β_0[k]+sst[1])).*(sst[2]-m_0[:,k])*
    end
    return m,β,W,ν,α
end

function updateR(Λ,m,α,ν,β) #variational expectation function
    r = Array{Float64}(undef,K,N) #responsibilities
    hat_π = Array{Float64}(undef,K)
    hat_Λ = Array{Float64}(undef,K)
    for k=1:K
        hat_Λ[k] = 1/2*(2*log(2)+logdet(Λ[:,:,k])+digamma(ν[k]/2)+digamma((ν[k]-1)/2))
        hat_π[k] = exp(digamma(α[k])-digamma(sum(α)))
        for n=1:N
           r[k,n] = hat_π[k]*exp(-hat_Λ[k]-1/β[k] - (ν[k]/2)*(X[:,n]-m[:,k])'*Λ[:,:,k]*(X[:,n]-m[:,k]))
        end
    end
    for n=1:N
        r[:,n] = r[:,n]./ sum(r[:,n]) #normalize to ensure r represents probabilities
    end
    return r
end

max_iter = 120
#store the inference results in these vectors
ν = fill!(Array{Float64}(undef,K,max_iter),3)
β = fill!(Array{Float64}(undef,K,max_iter),1.0)
α = fill!(Array{Float64}(undef,K,max_iter),0.01)
R = Array{Float64}(undef,K,N,max_iter)
M = Array{Float64}(undef,2,K,max_iter)
Λ = Array{Float64}(undef,2,2,K,max_iter)
clusters_vb = Array{Distribution}(undef,K,max_iter) #clusters to be plotted
#initialize prior distribution parameters
M[:,:,1] = rand(MersenneTwister(42), 2, K) .* [4, 50] .+ [1, 50]
for k=1:K
    Λ[:,:,k,1] = [1.0 0;0 0.01]
    R[k,:,1] = 1/(K)*ones(N)
    clusters_vb[k,1] = MvNormal(M[:,k,1],PDMats.PDMat(convert(Matrix,Hermitian(inv(ν[1,1].*Λ[:,:,k,1])))))
end
#variational inference
for i=1:max_iter-1
    #variational expectation
    R[:,:,i+1] = updateR(Λ[:,:,:,i],M[:,:,i],α[:,i],ν[:,i],β[:,i])
    #variational minimisation
    M[:,:,i+1],β[:,i+1],Λ[:,:,:,i+1],ν[:,i+1],α[:,i+1] = updateMeanPrecisionPi(M[:,:,i],β[:,i],Λ[:,:,:,i],ν[
```

```
    for k=1:K
        clusters_vb[k,i+1] = MvNormal(M[:,k,i+1],PDMats.PDMat(convert(Matrix,Hermitian(inv(ν[k,i+1].*Λ[:,:,k
    end
end

include("scripts/gmm_plot.jl") # Holds plotting function
plots = [plotGMM(X, clusters_vb[:,1], R[:,:,1], "Initial situation")]
for i=LinRange(2, 120, 5)
    i = round(Int,i)
    push!(plots, plotGMM(X, clusters_vb[:,i], R[:,:,i], "After $(i) iterations"))
end
plot(plots..., layout=(2,3), size=(1100, 600))
```

Out[2]:



- The generated figure looks much like Figure 10.6 in Bishop. The plots show FEM results for a GMM of $K = 6$ Gaussians applied to the Old Faithful data set. The ellipses denote the one standard-deviation density contours for each of the components, and the color coding of the data points reflects the "soft" class label assignments. Components whose expected mixing coefficient are numerically indistinguishable from zero are not plotted.

Variational Inference and The Method of Maximum Entropy

- We derived variational inference by substituting a variational posterior $q(z)$ for the Bayesian posterior $p(z|x)$ in the CA decomposition of (negative log) Bayesian evidence for a model. This is clever, but reveals nothing about the foundations of variational inference. Is variational inference any good?
- In Caticha (2010) (based on earlier work by Shore and Johnson (1980)), the **Method of Maximum (Relative) Entropy** is developed for rational updating of priors to posteriors when faced with new information in the form of constraints. Caticha's argumentation is as follows:
  - Consider prior beliefs (ie, a generative model) $p(x, z)$ about observed and latent variables x and z. Assume that new information in the form of (data, factorization or form) constraints is obtained and we are interested in the "best update" to a posterior $q(x, z)$.
  - We first establish that new observations of x can be phrased as constraints on the variational posterior q. For instance, a new observation $x_1 = 5$ can be formulated as a posterior constraint $q(x_1) = \delta(x_1 - 5)$.
  - In order to define what "best update" means, Caticha assumed a ranking function $S[q]$ that generates a preference score for each candidate posterior q for a given prior p. The best update from p to q is then identified as

$$q^* = \arg\max_q S[q]\,,$$
$$\text{subject to constraints.}$$

- Similarly to Cox' method for deriving Probability Theory from a set of sensical assumptions, Caticha then introduced the following axioms, based on a rational principle (the **principle of minimal updating**, see Caticha 2010), that the ranking function needs to adhere to:
  1. *Locality*: local information has local effects.
  2. *Coordinate invariance*: the system of coordinates carries no information.
  3. *Independence*: When systems are known to be independent, it should not matter whether they are treated separately or jointly.
- It turns out that these three criteria **uniquely identify the Relative Entropy** as the proper ranking function:

$$S[q] = -\sum_z q(x,z)\log\frac{q(x,z)}{p(x,z)}$$

- This procedure to find the variational posterior q is called the Method of Maximum (Relative) Entropy (MRE). Note that, since $S[q] = -F[q]$, constrained Relative Entropy maximization is equivalent to constrained Free Energy minimization!

- $\Rightarrow$ When information is supplied in the form of constraints on the posterior (such as form/factorization constraints and new observations as data constraints), we *should* select the posterior that minimizes the constrained Free Energy. **Constrained FE minimization is the proper method for inference!**

- Bayes rule is the global solution of constrained FEM when all constraints are data constraints, ie, delta distributions on $q(x)$. Hence, Bayes rule is a special case of constrained FEM. Bayes rule only applies to updating belief on the basis of new observations. FE minimization is the best inference method you can do under the given constraints.

Interesting Decompositions of the Free Energy Functional

- In rounding up this lession, we summarize a few interesting decompositions of the FE functional, making use of $p(x,z) = p(z|x)p(x) = p(x|z)p(z)$

$$\begin{aligned}
\mathrm{F}[q] &\triangleq \sum_z q(z)\log\frac{q(z)}{p(x,z)} \\
&= \underbrace{\sum_z q(z)\log\frac{1}{p(x,z)}}_{\text{energy}} - \underbrace{\sum_z q(z)\log\frac{1}{q(z)}}_{\text{entropy}} && \text{(EE)} \\
&= \underbrace{\sum_z q(z)\log\frac{q(z)}{p(z|x)}}_{\text{inference bound}\geq 0} - \underbrace{\log p(x)}_{\text{log-evidence}} && \text{(BE)} \\
&= \underbrace{\sum_z q(z)\log\frac{q(z)}{p(z)}}_{\text{complexity}} - \underbrace{\sum_z q(z)\log p(x|z)}_{\text{accuracy}} && \text{(CA)}
\end{aligned}$$

- These decompositions are very insightful and we will label them respectively as *energy-entropy* (EE), *bound-evidence* (BE), and *complexity-accuracy* (CA) decompositions.

- In the Bayesian Machine Learning lecture, we discussed the CA decomposition of Bayesian model evidence to support the interpretation of evidence as a model performance criterion. Here, we recognize that FE allows a similar CA decomposition: minimizing FE increases data fit and decreases model complexity. Hence, FE is a good model performance criterion.

- The CA decomposition makes use of the prior $p(z)$ and likelihood $p(x|z)$, both of which are selected by the engineer, so the FE can be evaluated with this decomposition!

- The BE decomposition restates what we derived earlier, namely that the FE is an upperbound on the (negative) log-evidence. The bound is the KL-divergence between the variational posterior $q(z)$ and the (perfect) Bayesian posterior $p(z|x)$. Global minimization of FE with only data constraints drives the KL-divergence to zero and results to perfect Bayesian inference.

- The BE decomposition can also be interpreted as problem representation costs (negative log-evidence) plus solution proposal costs (the KL-divergence bound), see the Intelligent Agent and Active Inference lesson (slide on Problem and Solution costs) for more details.

- The EE decomposition provides a link to the second law of thermodynamics: Minimizing FE leads to entropy maximization, subject to constraints, where in this case the constraints are imposed by the postulated generative model.

Variational Inference in Practice

- For most interesting models of real-world problems, Bayes rule is not tractible. Therefore, the usage of approximate variational Bayesian inference in real-world settings is rising rapidly.
- A toolbox such as RxInfer makes it possible to specify a complex model and automate the inference process by constrained Free Energy minimization.
- Note that model specification, even for complex models, usually does not take more than 1 page of code. As a result, you can, in principle, solve very complex problems by automated inference in a complex model with less than 1 page of code.
- $\Rightarrow$ Compared to writing an application algorithm of, say 40 pages of code, solving problems by automated variational inference is potentially a big deal for the future design of information processing systems.

OPTIONAL SLIDES

FE Minimization with Mean-field Factorization Constraints: the CAVI Approach

- Let's work out FE minimization with additional mean-field constraints (=full factorization) constraints:

$$q(z) = \prod_{j=1}^{m} q_j(z_j)\,.$$

  - In other words, the posteriors for z_j are all considered independent. This is a strong constraint but leads often to good solutions.
- Given the mean-field constraints, it is possible to derive the following expression for the optimal solutions $q_j^*(z_j)$, for $j = 1, \ldots, m$:

$$\boxed{\begin{aligned}\log q_j^*(z_j) &\propto \mathrm{E}_{q^*_{-j}}\left[\log p(x,z)\right] \\ &= \underbrace{\sum_{z_{-j}} q^*_{-j}(z_{-j}) \underbrace{\log p(x,z)}_{\text{"field"}}}_{\text{"mean field"}}\end{aligned}} \tag{B-10.9}$$

where we defined
$q^*_{-j}(z_{-j}) \triangleq q_1^*(z_1) q_2^*(z_2) \cdots q_{j-1}^*(z_{j-1}) q_{j+1}^*(z_{j+1})$
$\cdots q_m^*(z_m)$
.

- **Proof** (from Blei, 2017): We first rewrite the FE as a function of $q_j(z_j)$ only:

$$\begin{aligned} F[q_j] = \mathbb{E}_{q_j}\left[\mathbb{E}_{q_{-j}}\left[\log p(x, z_j, z_{-j})\right]\right] \\ - \mathbb{E}_{q_j}\left[\log q_j(z_j)\right] + \texttt{const.}\,, \end{aligned}$$

  where the constant holds all terms that do not depend on z_j. This expression can be written as

$$F[q_j] = \sum_{z_j} q_j(z_j) \log \frac{q_j(z_j)}{\exp\left(\mathbb{E}_{q_{-j}}\left[\log p(x, z_j, z_{-j})\right]\right)}$$

  which is a KL-divergence that is minimized by Eq. B-10.9. (end proof)
- This is not yet a full solution to the FE minimization task since the solution $q_j^*(z_j)$ depends on expectations that involve other solutions $q^*_{i\neq j}(z_{i\neq j})$, and each of these other solutions $q^*_{i\neq j}(z_{i\neq j})$ depends on an expection that involves $q_j^*(z_j)$.
- In practice, we solve this chicken-and-egg problem by an iterative approach: we first initialize all $q_j(z_j)$ (for $j = 1, \ldots, m$) to an appropriate initial distribution and then cycle through the factors in turn by solving eq.B-10.9 and update $q^*_{-j}(z_{-j})$ with the latest estimates. (See Blei, 2017, Algorithm 1, p864).
- This algorithm for approximating Bayesian inference is known **Coordinate Ascent Variational Inference** (CAVI).

FE Minimization by the Expectation-Maximization (EM) Algorithm

- The EM algorithm is a special case of FE minimization that focusses on Maximum-Likelihood estimation for models with latent variables.
- Consider a model

$$p(x, z, \theta)$$

with observations $x = \{x_n\}$, latent variables $z = \{z_n\}$ and parameters θ.

- We can write the following FE functional for this model:

$$F[q] = \sum_z \sum_\theta q(z, \theta) \log \frac{q(z, \theta)}{p(x, z, \theta)}$$

- The EM algorithm makes the following simplifying assumptions:

1. The prior for the parameters is uninformative (uniform). This implies that

$$p(x, z, \theta) = p(x, z|\theta)p(\theta) \propto p(x, z|\theta)$$

2. A factorization constraint

$$q(z, \theta) = q(z)q(\theta)$$

3. The posterior for the parameters is a delta function:

$$q(\theta) = \delta(\theta - \hat{\theta})$$

- Basically, these three assumptions turn FE minimization into maximum likelihood estimation for the parameters θ and the FE simplifies to

$$F[q, \theta] = \sum_z q(z) \log \frac{q(z)}{p(x, z|\theta)}$$

- The EM algorithm minimizes this FE by iterating (iteration counter: i) over

$$\begin{aligned} \mathcal{L}^{(i)}(\theta) &= \sum_z \overbrace{p(z|x, \theta^{(i-1)})}^{q^{(i)}(z)} \log p(x, z|\theta) && \text{the E-step} \\ \theta^{(i)} &= \arg\max_\theta \mathcal{L}^{(i)}(\theta) && \text{the M-step} \end{aligned}$$

- These choices are optimal for the given FE functional. In order to see this, consider the two decompositions

$$F[q, \theta] = \underbrace{-\sum_z q(z) \log p(x, z|\theta)}_{\text{energy}} \underbrace{- \sum_z q(z) \log \frac{1}{q(z)}}_{\text{entropy}} \qquad \text{(EE)}$$

$$= \underbrace{\sum_z q(z) \log \frac{q(z)}{p(z|x, \theta)}}_{\text{divergence}} \underbrace{- \log p(x|\theta)}_{\text{log-likelihood}} \qquad \text{(DE)}$$

- The DE decomposition shows that the FE is minimized for the choice $q(z) := p(z|x, \theta)$. Also, for this choice, the FE equals the (negative) log-evidence (, which is this case simplifies to the log-likelihood).
- The EE decomposition shows that the FE is minimized wrt θ by minimizing the energy term. The energy term is computed in the E-step and optimized in the M-step.
  - Note that in the EM literature, the energy term is often called the *expected complete-data log-likelihood*.)
- In order to execute the EM algorithm, it is assumed that we can analytically execute the E- and M-steps. For a large set of models (including models whose distributions belong to the exponential family of distributions), this is indeed the case and hence the large popularity of the EM algorithm.
- The EM algorihm imposes rather severe assumptions on the FE (basically approximating Bayesian inference by maximum likelihood estimation). Over the past few years, the rise of Probabilistic Programming languages has dramatically increased the range of models for which the parameters can by estimated autmatically by (approximate) Bayesian inference, so the popularity of EM is slowly waning. (More

on this in the Probabilistic Programming lessons).

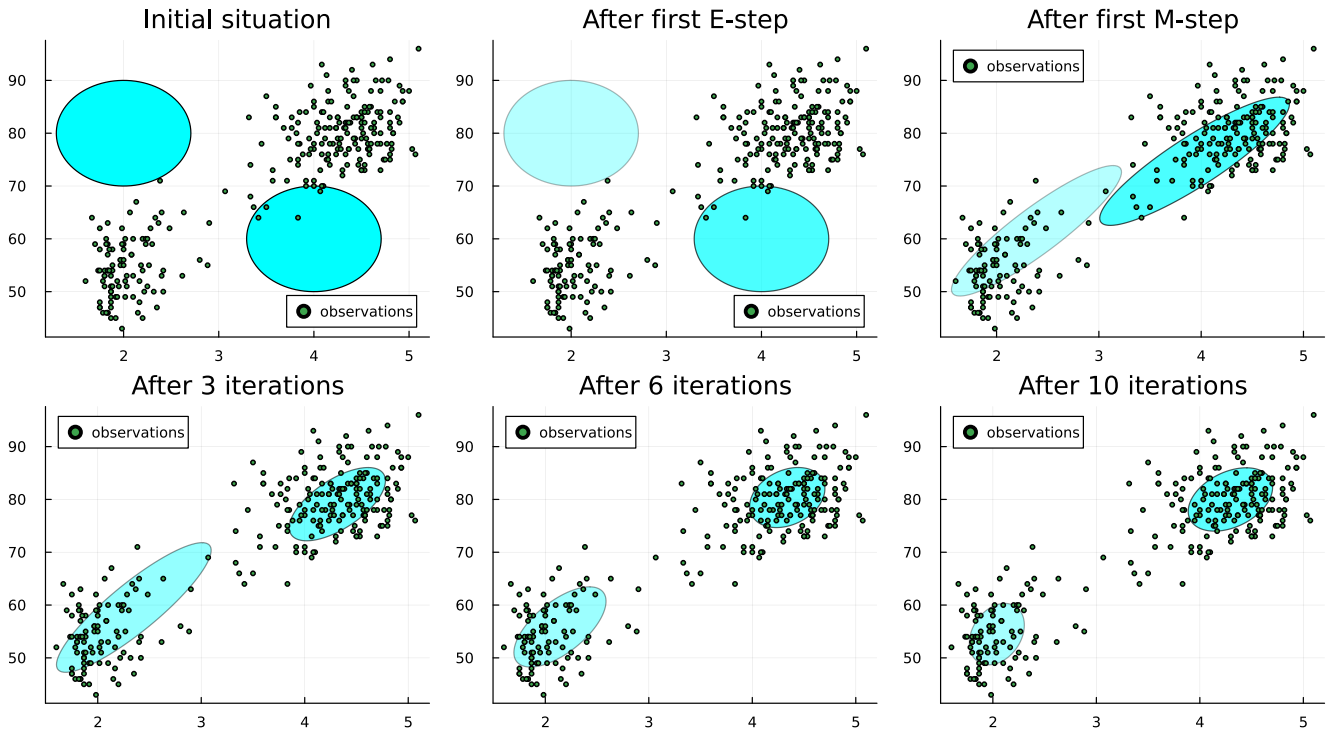
- Bishop (2006) works out EM for the GMM in section 9.2.

Code Example: EM-algorithm for the GMM on the Old-Faithful data set

We'll perform clustering on the data set from the illustrative example by fitting a GMM consisting of two Gaussians using the EM algorithm.

In [3]:
```
using DataFrames, CSV, LinearAlgebra
include("scripts/gmm_plot.jl") # Holds plotting function
old_faithful = CSV.read("datasets/old_faithful.csv", DataFrame);

X =  Array(Matrix{Float64}(old_faithful)')
N = size(X, 2)

# Initialize the GMM. We assume 2 clusters.
clusters = [MvNormal([4.;60.], [.5 0;0 10^2]);
            MvNormal([2.;80.], [.5 0;0 10^2])];
π_hat = [0.5; 0.5]                    # Mixing weights
γ = fill!(Matrix{Float64}(undef,2,N), NaN)  # Responsibilities (row per cluster)

# Define functions for updating the parameters and responsibilities
function updateResponsibilities!(X, clusters, π_hat, γ)
    # Expectation step: update γ
    norm = [pdf(clusters[1], X) pdf(clusters[2], X)] * π_hat
    γ[1,:] = (π_hat[1] * pdf(clusters[1],X) ./ norm)'
    γ[2,:] = 1 .- γ[1,:]
end
function updateParameters!(X, clusters, π_hat, γ)
    # Maximization step: update π_hat and clusters using ML estimation
    m = sum(γ, dims=2)
    π_hat = m / N
    μ_hat = (X * γ') ./ m'
    for k=1:2
        Z = (X .- μ_hat[:,k])
        Σ_k = Symmetric(((Z .* (γ[k,:])') * Z') / m[k])
        clusters[k] = MvNormal(μ_hat[:,k], convert(Matrix, Σ_k))
    end
end

# Execute the algorithm: iteratively update parameters and responsibilities
plots = [plotGMM(X, clusters, γ, "Initial situation")]
updateResponsibilities!(X, clusters, π_hat, γ)
push!(plots, plotGMM(X, clusters, γ, "After first E-step"))
updateParameters!(X, clusters, π_hat, γ)
push!(plots, plotGMM(X, clusters, γ, "After first M-step"))
iter_counter = 1
for i=1:3
    for j=1:i+1
        updateResponsibilities!(X, clusters, π_hat, γ)
        updateParameters!(X, clusters, π_hat, γ)
        iter_counter += 1
    end
    push!(plots, plotGMM(X, clusters, γ, "After $(iter_counter) iterations"))
end
plot(plots..., layout=(2,3), size=(1100, 600))
```

Out[3]:



Message Passing for Free Energy Minimization

- The Sum-Product (SP) update rule implements perfect Bayesian inference.
- Sometimes, the SP update rule is not analytically solvable.
- Fortunately, for many well-known Bayesian approximation methods, a message passing update rule can be created, e.g. Variational Message Passing (VMP) for variational inference.
- In general, all of these message passing algorithms can be interpreted as minimization of a constrained free energy (e.g., see Senoz et al. (2021), and hence these message passing schemes comply with Caticha's Method of Maximum Relative Entropy, which, as discussed in the variational Bayes lesson is the proper way for updating beliefs.
- Different message passing updates rules can be combined to get a hybrid inference method in one model.

The Local Free Energy in a Factor Graph

- Consider an edge x_j in a Forney-style factor graph for a generative model $p(x) = p(x_1, x_2, \ldots, x_N)$.
- Assume that the graph structure (factorization) is specified by

$$p(x) = \prod_{a=1}^{M} p_a(x_a)$$

  where a is a set of indices.
- Also, we assume a mean-field approximation for the posterior:

$$q(x) = \prod_{i=1}^{N} q_i(x_i)$$

  and consequently a corresponding free energy functional

$$
\begin{aligned}
F[q] &= \sum_x q(x) \log \frac{q(x)}{p(x)} \\
&= \sum_i \sum_{x_i} \left(\prod_{i=1}^N q_i(x_i) \right) \log \frac{\prod_{i=1}^N q_i(x_i)}{\prod_{a=1}^M p_a(x_a)}
\end{aligned}
$$

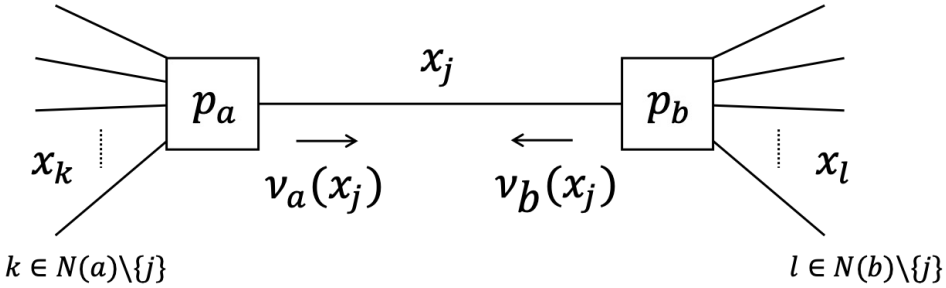
- With these assumptions, it can be shown that the FE evaluates to (exercise)

$$
F[q] = \underbrace{\sum_{a=1}^M \sum_{x_a} \left(\prod_{j \in N(a)} q_j(x_j) \cdot (-\log p_a(x_a)) \right)}_{\text{node energy } U[p_a]} - \underbrace{\sum_{i=1}^N \sum_{x_i} q_i(x_i) \log \frac{1}{q_i(x_i)}}_{\text{edge entropy } H[q_i]}
$$

- In words, the FE decomposes into a sum of (expected) energies for the nodes minus the entropies on the edges.

Variational Message Passing

- Let us now consider the local free energy that is associated with edge corresponding to x_j.



- Apparently (see previous slide), there are three contributions to the free energy for x_j:
  - one entropy term for the edge x_j
  - two energy terms: one for each node that attaches to x_j (in the figure: nodes p_a and p_b)
- The local free energy for x_j can be written as (exercise)

$$
F[q_j] \propto \sum_{x_j} q(x_j) \log \frac{q_j(x_j)}{\nu_a(x_j) \cdot \nu_b(x_j)}
$$

  where

$$
\begin{aligned}
\nu_a(x_j) &\propto \exp(\mathbb{E}_{q_k}\left[\log p_a(x_a)\right]) \\
\nu_b(x_j) &\propto \exp(\mathbb{E}_{q_l}\left[\log p_b(x_b)\right])
\end{aligned}
$$

  and $\mathbb{E}_{q_k}\left[\cdot\right]$ is an expectation w.r.t. all $q(x_k)$ with $k \in N(a) \setminus j$.

- $\nu_a(x_j)$ and $\nu_b(x_j)$ can be locally computed in nodes a and b respectively and can be interpreted as colliding messages over edge x_j.
- Local free energy minimization is achieved by setting

$$
q_j(x_j) \propto \nu_a(x_j) \cdot \nu_b(x_j)
$$

- Note that message $\nu_a(x_j)$ depends on posterior beliefs over incoming edges (k) for node a, and in turn, the message from node a towards edge x_k depends on the belief $q_j(x_j)$. I.o.w., direct mutual dependencies exist between posterior beliefs over edges that attach to the same node.
- These considerations lead to the Variational Message Passing procedure, which is an iterative free energy minimization procedure that can be executed completely through locally computable messages.

- Procedure VMP, see Dauwels (2007), section 3

    1. Initialize all messages q and ν, e.g., $q(\cdot) \propto 1$ and $\nu(\cdot) \propto 1$.
    2. Select an edge z_k in the factor graph of $f(z_1, \ldots, z_m)$.
    3. Compute the two messages $\overrightarrow{\nu}(z_k)$ and $\overleftarrow{\nu}(z_k)$ by applying the following generic rule:

    $$\overrightarrow{\nu}(y) \propto \exp \left(\mathbb{E}_q \left[\log g(x_1, \ldots, x_n, y) \right] \right)$$

    4. Compute the marginal $q(z_k)$

    $$q(z_k) \propto \overrightarrow{\nu}(z_k) \overleftarrow{\nu}(z_k)$$

    and send it to the two nodes connected to the edge x_k.
    5. Iterate 2–4 until convergence.

The Bethe Free Energy and Belief Propagation

- We showed that, under mean field assumptions, the FE can be decomposed into a sum of local FE contributions for the nodes (a) and edges (i):

$$F[q] = \underbrace{\sum_{a=1}^{M} \sum_{x_a} \left(\prod_{j \in N(a)} q_j(x_j) \cdot (-\log p_a(x_a)) \right)}_{\text{node energy } U[p_a]} - \underbrace{\sum_{i=1}^{N} \sum_{x_i} q_i(x_i) \log \frac{1}{q_i(x_i)}}_{\text{edge entropy } H[q_i]}$$

- The mean field assumption is very strong and may lead to large inference costs ($\mathrm{KL}(q(x), p(x|\mathrm{data}))$). A more relaxed assumption is to allow joint posterior beliefs over the variables that attach to a node. This idea is expressed by the Bethe Free Energy:

$$F_B[q] = \sum_{a=1}^{M} \left(\sum_{x_a} q_a(x_a) \log \frac{q_a(x_a)}{p_a(x_a)} \right) - \sum_{i=1}^{N} (d_i - 1) \sum_{x_i} q_i(x_i) \log q_i(x_i)$$

  where $q_a(x_a)$ is the posterior joint belief over the variables x_a (i.e., the set of variables that attach to node a), $q_i(x_i)$ is the posterior marginal belief over the variable x_i and d_i is the number of factor nodes that link to edge i. Moreover, $q_a(x_a)$ and $q_i(x_i)$ are constrained to obey the following equalities:

$$\sum_{x_a \setminus x_i} q_a(x_a) = q_i(x_i), \quad \forall i, \forall a$$

$$\sum_{x_i} q_i(x_i) = 1, \quad \forall i$$

$$\sum_{x_a} q_a(x_a) = 1, \quad \forall a$$

- We form the Lagrangian by augmenting the Bethe Free Energy functional with the constraints:

$$L[q] = F_B[q] + \sum_i$$
$$\sum_{a \in N(i)} \lambda_{ai}(x_i) \left(q_i(x_i) - \sum_{x_a \setminus x_i} q(x_a) \right) +$$
$$\sum_i \gamma_i \left(\sum_{x_i} q_i(x_i) - 1 \right) +$$
$$\sum_a \gamma_a \left(\sum_{x_a} q_a(x_a) - 1 \right)$$

- The stationary solutions for this Lagrangian are given by

$$q_a(x_a) = f_a(x_a) \exp$$
$$\left(\gamma_a - 1 + \sum_{i \in N(a)} \lambda_{ai}(x_i) \right)$$
$$q_i(x_i) = \exp \left(1 - \gamma_i + \sum_{a \in N(i)} \lambda_{ai}(x_i) \right)^{\frac{1}{d_i - 1}}$$

  where $N(i)$ denotes the factor nodes that have x_i in their arguments and $N(a)$ denotes the set of variables in the argument of f_a.

- Stationary solutions are functions of Lagrange multipliers. This means that Lagrange multipliers need to be determined. Lagrange multipliers can be determined by plugging the stationary solutions back into the constraint specification and solving for the multipliers which ensure that the constraint is satisfied. The first constraint we consider is normalization, which yields the following identification:

$$\gamma_a = 1 - \log$$
$$\left(\sum_{x_a} f_a(x_a) \exp \left(\sum_{i \in N(a)} \lambda_{ai}(x_i) \right) \right)$$
$$\gamma_i = 1 + (d_i - 1) \log$$
$$\left(\sum_{x_i} \exp \left(\frac{1}{d_i - 1} \sum_{a \in N(i)} \lambda_{ai}(x_i) \right) \right).$$

- The functional form of the Lagrange multipliers that corresponds to the normalization constraint enforces us to obtain the Lagrange multipliers that correspond to the marginalization constraint. To do so we solve for

$$\sum_{x_a \setminus x_i} f_a(x_a) \exp\left(\sum_{i \in N(a)} \lambda_{ai}(x_i)\right) = \exp$$

$$\left(\sum_{a \in N(i)} \lambda_{ai}(x_i)\right)^{\frac{1}{d_i - 1}}$$

$$\exp(\lambda_{ai}(x_i)) \sum_{x_a \setminus x_i} f_a(x_a) \exp$$

$$\left(\sum_{\substack{j \in N(a) \\ j \neq i}} \lambda_{aj}(x_j)\right) = \exp$$

$$\left(\sum_{a \in N(i)} \lambda_{ai}(x_i)\right)^{\frac{1}{d_i - 1}}$$

$$\exp(\lambda_{ai}(x_i) + \lambda_{ia}(x_i)) = \exp$$

$$\left(\sum_{a \in N(i)} \lambda_{ai}(x_i)\right)^{\frac{1}{d_i - 1}},$$

where we defined an auxilary function

$$\exp(\lambda_{ia}(x_i)) \triangleq \sum_{x_a \setminus x_i} f_a(x_a) \exp$$

$$\left(\sum_{\substack{j \in N(a) \\ j \neq i}} \lambda_{aj}(x_j)\right).$$

This definition is valid since it can be inverted by the relation

$$\lambda_{ia}(x_i) = \frac{2 - d_i}{d_i - 1} \lambda_{ai}(x_i) + \frac{1}{d_i - 1} \sum_{\substack{c \in N(i) \\ c \neq a}} \lambda_{ci}(x_i)$$

- In general it is not possible to solve for the Lagrange multipliers analytically and we resort to iteratively obtaining the solutions. This leads to the **Belief Propagation algorithm** where the exponentiated Lagrange multipliers (messages) are updated iteratively via

$$\mu_{ia}^{(k+1)}(x_i) = \sum_{x_a \setminus x_i} f_a(x_a) \prod_{\substack{j \in N(a) \\ j \neq i}} \mu_{aj}^{(k)}(x_j)$$

$$\mu_{ai}^{(k)}(x_i) = \prod_{\substack{c \in N(i) \\ c \neq a}} \mu_{ic}^{(k)}(x_i),$$

where k denotes iteration number and the messages are defined as

$$\mu_{ia}(x_i) \triangleq \exp(\lambda_{ia}(x_i))$$
$$\mu_{ai}(x_i) \triangleq \exp(\lambda_{ai}(x_i)).$$

- For a more complete overview of message passing as Bethe Free Energy minimization, see Senoz et al. (2021).

Dynamic Models

Preliminaries

- Goal
  - Introduction to dynamic (=temporal) Latent Variable Models, including the Hidden Markov Model and Kalman filter.
- Materials
  - Mandatory
    - These lecture notes
  - Optional
    - Bishop pp.605-615 on Hidden Markov Models
    - Bishop pp.635-641 on Kalman filters
    - Faragher (2012), Understanding the Basis of the Kalman Filter
    - Minka (1999), From Hidden Markov Models to Linear Dynamical Systems

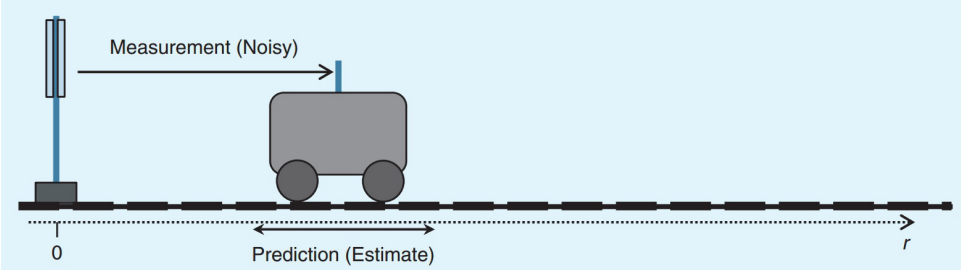
Example Problem

- We consider a one-dimensional cart position tracking problem, see Faragher (2012).
- The hidden states are the position z_t and velocity $\dot{z}_t$. We can apply an external acceleration/breaking force u_t. (Noisy) observations are represented by x_t.
- The equations of motions are given by

$$\begin{bmatrix} z_t \\ \dot{z}_t \end{bmatrix} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} z_{t-1} \\ \dot{z}_{t-1} \end{bmatrix} + \begin{bmatrix} (\Delta t)^2/2 \\ \Delta t \end{bmatrix} u_t + \mathcal{N}(0, \Sigma_z)$$

$$x_t = \begin{bmatrix} z_t \\ \dot{z}_t \end{bmatrix} + \mathcal{N}(0, \Sigma_x)$$

- Task: Infer the position z_t after 10 time steps. (Solution later in this lesson).



Dynamical Models

- In this lesson, we consider models where the sequence order of observations matters.
- Consider the *ordered* observation sequence $x^T \triangleq (x_1, x_2, \ldots, x_T)$.
  - (For brevity, in this lesson we use the notation x_t^T to denote $(x_t, x_{t+1}, \ldots, x_T)$ and drop the subscript if $t = 1$, so $x^T = x_1^T = (x_1, x_2, \ldots, x_T)$).
- We wish to develop a generative model

$$p(x^T)$$

  that 'explains' the time series x^T.
- We cannot use the IID assumption $p(x^T) = \prod_t p(x_t)$ for a time series, since consecutive observations may have statistical dependencies. In general, we can *always* use the **chain rule** (a.k.a. **the general product rule**)

$$\begin{aligned} p(x^T) &= p(x_T|x^{T-1})\, p(x^{T-1}) \\ &= p(x_T|x^{T-1})\, p(x_{T-1}|x^{T-2}) \cdots p(x_2|x_1)\, p(x_1) \\ &= p(x_1) \prod_{t=2}^{T} p(x_t \mid x^{t-1}) \end{aligned}$$

which is true for any model $p(x^T)$.

- Generally, we will want to limit the depth of dependencies on previous observations. For example, a K-th order linear **Auto-Regressive** (AR) model is given by

$$p(x_t \,|\, x^{t-1}) = \mathcal{N}\left(x_t \,\middle|\, \sum_{k=1}^{K} a_k x_{t-k}\,, \sigma^2\right)$$

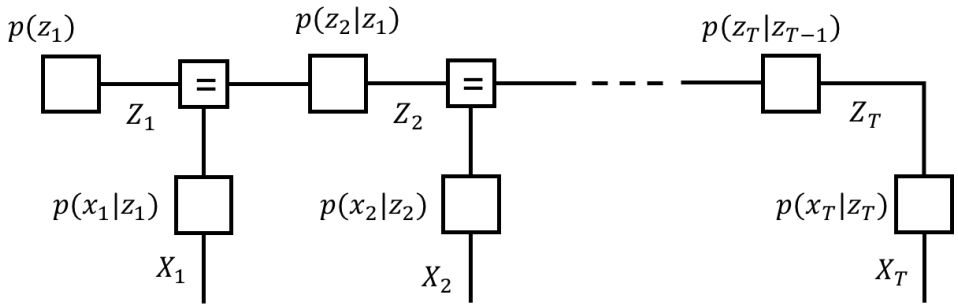
  limits the dependencies to the past K samples.

State-space Models

- A limitation of AR models is that they need a lot of parameters in order to create a flexible model. E.g., if $x_t \in \mathbb{R}^M$ is an M-dimensional time series, then the K-th order AR model for x_t will have KM^2 parameters.
- Similar to our work on Gaussian Mixture models, we can create a flexible dynamic system by introducing *latent* (unobserved) variables $z^T \triangleq (z_1, z_2, \ldots, z_T)$ (one z_t for each observation x_t). In dynamic systems, observation-dependent latent variables z_t are called *state variables*.
- A **state space model** is a particular latent variable dynamical model defined by

$$p(x^T, z^T) = \underbrace{p(z_1)}_{\text{initial state}} \prod_{t=2}^{T} \underbrace{p(z_t \,|\, z_{t-1})}_{\text{state transitions}} \prod_{t=1}^{T} \underbrace{p(x_t \,|\, z_t)}_{\text{observations}}$$

  - The condition $p(z_t \,|\, z^{t-1}) = p(z_t \,|\, z_{t-1})$ is called a 1-st order Markov condition.
- The Forney-style factor graph for a state-space model:



Hidden Markov Models and Linear Dynamical Systems

- A **Hidden Markov Model** (HMM) is a specific state-space model with discrete-valued state variables z_t.
- Typically, z_t is a K-dimensional one-hot coded latent 'class indicator' with transition probabilities $a_{jk} \triangleq p(z_{tk} = 1 \,|\, z_{t-1,j} = 1)$, or equivalently,

$$p(z_t|z_{t-1}) = \prod_{k=1}^{K} \prod_{j=1}^{K} a_{jk}^{z_{t-1,j} \cdot z_{tk}}$$

  which is usually accompanied by an initial state distribution $p(z_{1k} = 1) = \pi_k$.
- The classical HMM has also discrete-valued observations but in pratice any (probabilistic) observation model $p(x_t|z_t)$ may be coupled to the hidden Markov chain.
- Another well-known state-space model with continuous-valued state variables z_t is the **(Linear) Gaussian Dynamical System** (LGDS), which is defined as

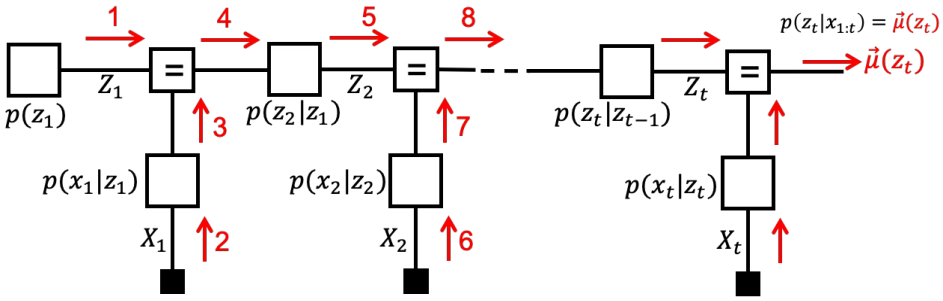
$$\begin{aligned} p(z_t \,|\, z_{t-1}) &= \mathcal{N}\left(z_t \,|\, A z_{t-1},\, \Sigma_z\right) \\ p(x_t \,|\, z_t) &= \mathcal{N}\left(x_t \,|\, C z_t,\, \Sigma_x\right) \\ p(z_1) &= \mathcal{N}\left(z_1 \,|\, \mu_1,\, \Sigma_1\right) \end{aligned}$$

- Note that the joint distribution over all states and observations $\{(x_1, z_1), \ldots, (x_t, z_t)\}$ is a (large-dimensional) Gaussian distribution. This means that, in principle, every inference problem on the LGDS model also leads to a Gaussian distribution.

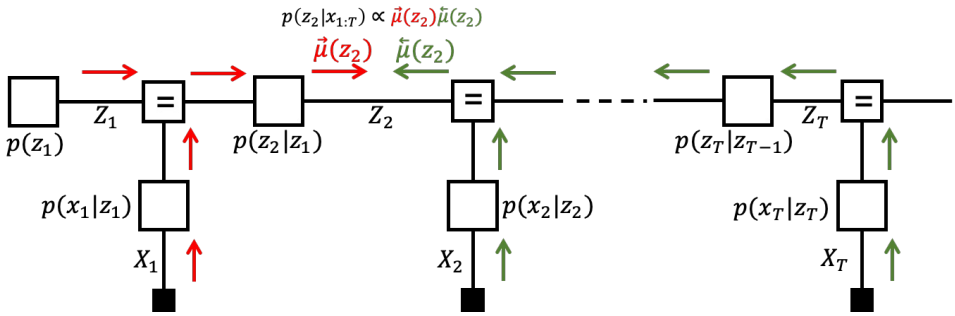
- HMM's and LGDS's (and variants thereof) are at the basis of a wide range of complex information processing systems, such as speech and language recognition, robotics and automatic car navigation, and even processing of DNA sequences.

Common Signal Processing Tasks as Message Passing-based Inference

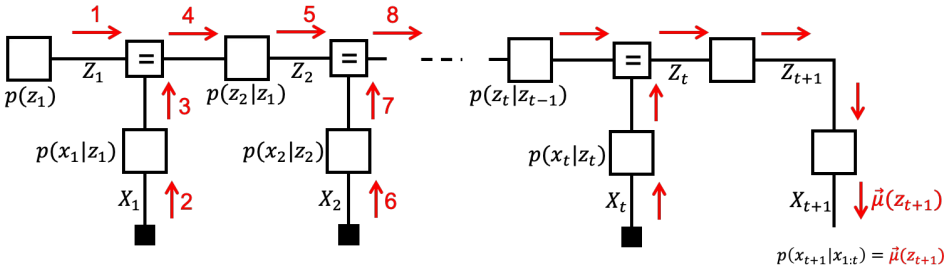
- As we have seen, inference tasks in linear Gaussian state space models can be analytically solved.
- However, these derivations quickly become cumbersome and prone to errors.
- Alternatively, we could specify the generative model in a (Forney-style) factor graph and use automated message passing to infer the posterior over the hidden variables. Here follows some examples.
- **Filtering**, a.k.a. state estimation: estimation of a state (at time step t), based on past and current (at t) observations.



- **Smoothing**: estimation of a state based on both past and future observations. Needs backward messages from the future.



- **Prediction**: estimation of future state or observation based only on observations of the past.



An Analytical Derivation of the Kalman Filter

- Let's work out the Kalman filter for a scalar linear Gaussian dynamical system:

$$
\begin{aligned}
p(z_t \mid z_{t-1}) &= \mathcal{N}(z_t \mid a z_{t-1}, \sigma_z^2) && \text{(state transition)} \\
p(x_t \mid z_t) &= \mathcal{N}(x_t \mid c z_t, \sigma_x^2) && \text{(observation)}
\end{aligned}
$$

- Technically, a **Kalman filter** is the solution to the recursive estimation (inference) of the hidden state z_t based on past observations in an LGDS, i.e., Kalman filtering solves the problem $p(z_t \mid x^t)$ based on the previous state estimate $p(z_{t-1} \mid x^{t-1})$ (the "prior") and a new observation x_t (in the context of the given model specification of course).

- In Bayesian terms, a Kalman filter computes

$$\begin{aligned}\overbrace{p(z_t \mid x^t)}^{\text{posterior}} \cdot \overbrace{p(x_t \mid x^{t-1})}^{\text{evidence}} &= p(x_t, z_t \mid x^{t-1}) \\ &= p(x_t \mid z_t) \cdot p(z_t \mid x^{t-1}) \\ &= p(x_t \mid z_t) \int p(z_t, z_{t-1} \mid x^{t-1}) \mathrm{d}z_{t-1} \\ &= \underbrace{p(x_t \mid z_t)}_{\text{likelihood}} \int \underbrace{p(z_t \mid z_{t-1})}_{\text{state transition}} \underbrace{p(z_{t-1} \mid x^{t-1})}_{\text{prior}} \mathrm{d}z_{t-1} \\ &= \mathcal{N}(x_t | c z_t, \sigma_x^2) \int \mathcal{N}(z_t \mid a z_{t-1}, \sigma_z^2) \\ &\quad \mathcal{N}(z_{t-1} \mid \mu_{t-1}, \sigma_{t-1}^2) \mathrm{d}z_{t-1} \qquad \text{(KF-1)}\end{aligned}$$

  where we assumed that the previous state estimate is given by

$$p(z_{t-1} \mid x^{t-1}) = \mathcal{N}(z_{t-1} \mid \mu_{t-1}, \sigma_{t-1}^2) \\ \text{(prior)}$$

- The result (Eq. KF-1) can be further worked out to analytical updates for the evidence and posterior. In the following, we often run into Gaussians of the form $\mathcal{N}(x \mid cz, \sigma^2)$ that we need to rewrite as an argument of z. We wil use the following "mean transformation" equality:

$$\mathcal{N}(x \mid cz, \sigma^2) = \frac{1}{c} \mathcal{N}\left(z \middle| \frac{x}{c}, \left(\frac{\sigma}{c}\right)^2\right) .$$

- Let's further work out the Kalman filter, starting from Eq. KF-1:

$$\underbrace{\mathcal{N}(x_t|cz_t,\sigma_x^2)}_{\text{likelihood}} \int \underbrace{\mathcal{N}(z_t \mid az_{t-1},\sigma_z^2)}_{\substack{\text{state transition}\\ \text{(use mean transformation)}}} \underbrace{\mathcal{N}(z_{t-1} \mid \mu_{t-1},\sigma_{t-1}^2)}_{\text{prior}} \mathrm{d}z_{t-1} =$$

$$= \mathcal{N}(x_t|cz_t,\sigma_x^2) \int \frac{1}{a} \underbrace{\mathcal{N}\left(z_{t-1} \mid \frac{z_t}{a}, \left(\frac{\sigma_z}{a}\right)^2\right) \mathcal{N}(z_{t-1} \mid \mu_{t-1},\sigma_{t-1}^2)}_{\text{use Gaussian multiplication formula SRG-6}} \mathrm{d}z_{t-1}$$

$$= \frac{1}{a}\mathcal{N}(x_t|cz_t,\sigma_x^2) \int \underbrace{\mathcal{N}\left(\mu_{t-1} \mid \frac{z_t}{a}, \left(\frac{\sigma_z}{a}\right)^2 + \sigma_{t-1}^2\right)}_{\text{not a function of } z_{t-1}} \underbrace{\mathcal{N}(z_{t-1} \mid \cdot,\cdot)}_{\text{integrates to 1}} \mathrm{d}z_{t-1}$$

$$= \frac{1}{a} \underbrace{\mathcal{N}(x_t|cz_t,\sigma_x^2)}_{\text{use mean transformation}} \underbrace{\mathcal{N}\left(\mu_{t-1} \mid \frac{z_t}{a}, \left(\frac{\sigma_z}{a}\right)^2 + \sigma_{t-1}^2\right)}_{\text{use mean transformation}}$$

$$= \frac{1}{c}\mathcal{N}\left(z_t \mid \frac{x_t}{c}, \left(\frac{\sigma_x}{c}\right)^2\right) \underbrace{\mathcal{N}\left(z_t \mid a\mu_{t-1}, \sigma_z^2 + (a\sigma_{t-1})^2\right)}_{\text{use SRG-6 again}}$$

$$= \underbrace{\frac{1}{c}\mathcal{N}\left(\frac{x_t}{c} \mid a\mu_{t-1}, \left(\frac{\sigma_x}{c}\right)^2 + \sigma_z^2 + (a\sigma_{t-1})^2\right)}_{\text{use mean transformation}} \mathcal{N}\left(z_t \mid \mu_t, \sigma_t^2\right)$$

$$= \underbrace{\mathcal{N}\left(x_t \mid ca\mu_{t-1}, \sigma_x^2 + c^2(\sigma_z^2 + a^2\sigma_{t-1}^2)\right)}_{\text{evidence } p(x_t|x^{t-1})} \cdot \underbrace{\mathcal{N}\left(z_t \mid \mu_t, \sigma_t^2\right)}_{\text{posterior } p(z_t|x^t)}$$

where the posterior mean μ_t and posterior variance σ_t^2 can be evaluated as follows:

$$\begin{aligned}
\rho_t^2 &= a^2\sigma_{t-1}^2 + \sigma_z^2 && \text{(predicted variance)}\\
K_t &= \frac{c\rho_t^2}{c^2\rho_t^2 + \sigma_x^2} && \text{(Kalman gain)}\\
\mu_t &= a\mu_{t-1} + K_t \cdot (x_t - ca\mu_{t-1}) && \text{(posterior mean)}\\
\sigma_t^2 &= (1 - c \cdot K_t)\,\rho_t^2 && \text{(posterior variance)}
\end{aligned}$$

- Kalman filtering consists of computing/updating these last four equations for each new observation (x_t). This is a very efficient recursive algorithm to estimate the state z_t from all observations (until t).
- The above derivation also evaluates the "instant" evidence

$$p(x_t|x^{t-1}) = \mathcal{N}\left(x_t \mid ca\mu_{t-1}, \sigma_x^2 + c^2(\sigma_z^2 + a^2\sigma_{t-1}^2)\right)$$

- Note that, for observed x^t, the evidence $p(x_t|x^{t-1})$ is a scalar number that scores how well the model predicts x^t, based on past observations x^{t-1}.
- Exam guide: the above derivation of the Kalman filter is too long and error-prone to be asked at an exam. You should be able to follow the derivation in detail, but will not be requested to reproduce the full derivation without some guidance. The complexity of the derivation underlines why inference should be automated by a toolbox (like RxInfer).

Multi-dimensional Kalman Filtering

- The Kalman filter equations can also be derived for multidimensional state-space models. In particular, for the model

$$\begin{aligned} z_t &= Az_{t-1} + \mathcal{N}(0,\Gamma) \\ x_t &= Cz_t + \mathcal{N}(0,\Sigma) \end{aligned}$$

the Kalman filter update equations for the posterior $p(z_t|x^t) = \mathcal{N}\left(z_t \mid \mu_t, V_t\right)$ are given by (see Bishop, pg.639)

$$\begin{aligned} P_t &= AV_{t-1}A^T + \Gamma && \text{(predicted variance)} \\ K_t &= P_t C^T \cdot \left(CP_tC^T + \Sigma\right)^{-1} && \text{(Kalman gain)} \\ \mu_t &= A\mu_{t-1} + K_t \cdot (x_t - CA\mu_{t-1}) && \text{(posterior state mean)} \\ V_t &= (I - K_tC)\,P_t && \text{(posterior state variance)} \end{aligned}$$

Code Example: Kalman Filtering and the Cart Position Tracking Example Revisited

- We can now solve the cart tracking problem of the introductory example by implementing the Kalman filter.

In [1]:
```
using Pkg; Pkg.activate("../."); Pkg.instantiate();
using IJulia; try IJulia.clear_output(); catch _ end
```

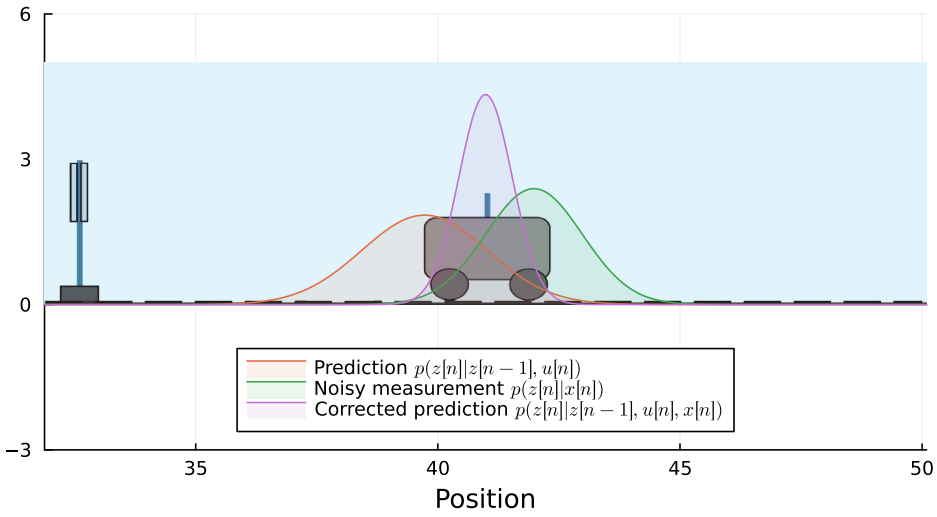
Out[1]:0

In [2]:
```
using RxInfer, LinearAlgebra, Plots

include("scripts/cart_tracking_helpers.jl")

# Specify the model parameters
Δt = 1.0                                          # assume the time steps to be equal in size
A = [1.0 Δt;
     0.0 1.0]
b = [0.5*Δt^2; Δt]
Σz = convert(Matrix,Diagonal([0.2*Δt; 0.1*Δt]))   # process noise covariance
Σx = convert(Matrix,Diagonal([1.0; 2.0]))         # observation noise covariance;

# Generate noisy observations
n = 10                                            # perform 10 timesteps
z_start = [10.0; 2.0]                             # initial state
u = 0.2 * ones(n)                                 # constant input u
noisy_x = generateNoisyMeasurements(z_start, u, A, b, Σz, Σx);

m_z = noisy_x[1]                                  # initial predictive mean
V_z = A * (1e8*Diagonal(I,2) * A') + Σz           # initial predictive covariance

for t = 2:n
    global m_z, V_z, m_pred_z, V_pred_z

    #predict
    m_pred_z = A * m_z + b * u[t]                 # predictive mean
    V_pred_z = A * V_z * A' + Σz                  # predictive covariance

    #update
    gain = V_pred_z * inv(V_pred_z + Σx)          # Kalman gain
    m_z = m_pred_z + gain * (noisy_x[t] - m_pred_z) # posterior mean update
    V_z = (Diagonal(I,2)-gain)*V_pred_z           # posterior covariance update
end

println("Prediction: ",MvNormalMeanCovariance(m_pred_z,V_pred_z))
println("Measurement: ", MvNormalMeanCovariance(noisy_x[n],Σx))
println("Posterior: ", MvNormalMeanCovariance(m_z,V_z))
plotCartPrediction(m_pred_z[1], V_pred_z[1], m_z[1], V_z[1], noisy_x[n][1], Σx[1][1])
```

```
Prediction: MvNormalMeanCovariance(
μ: [39.725469627517846, 3.9185105662333326]
```

```
Σ: [1.2958787328575079 0.3921572953097835; 0.3921572953130242 0.3415636711134632]
)

Measurement: MvNormalMeanCovariance(
μ: [41.98403629603774, 4.105464041246935]
Σ: [1.0 0.0; 0.0 2.0]
)

Posterior: MvNormalMeanCovariance(
μ: [40.98535689913251, 4.280292459270916]
Σ: [0.5516100293973586 0.15018972175285758; 0.15018972175409862 0.24143326063188655]
)
```

Out[2]:



The Cart Tracking Problem Revisited: Inference by Message Passing

- Let's solve the cart tracking problem by sum-product message passing in a factor graph like the one depicted above. All we have to do is create factor nodes for the state-transition model $p(z_t|z_{t-1})$ and the observation model $p(x_t|z_t)$. Then we let RxInfer execute the message passing schedule.

In [3]:
```
@model function cart_tracking(x, n, A, B, Σz, Σx, z_prev_m_0, z_prev_v_0, u)

    local z

    z_prior ~ MvNormalMeanCovariance(z_prev_m_0, z_prev_v_0)

    z_prev = z_prior
    for i in 1:n

        z[i] ~ MvNormalMeanCovariance(A*z_prev + B*u[i], Σz)
        x[i] ~ MvNormalMeanCovariance(z[i], Σx)

        z_prev = z[i]
    end
    return (z,)
end
```

Now that we've built the model, we can perform Kalman filtering by inserting measurement data into the model and performing message passing.

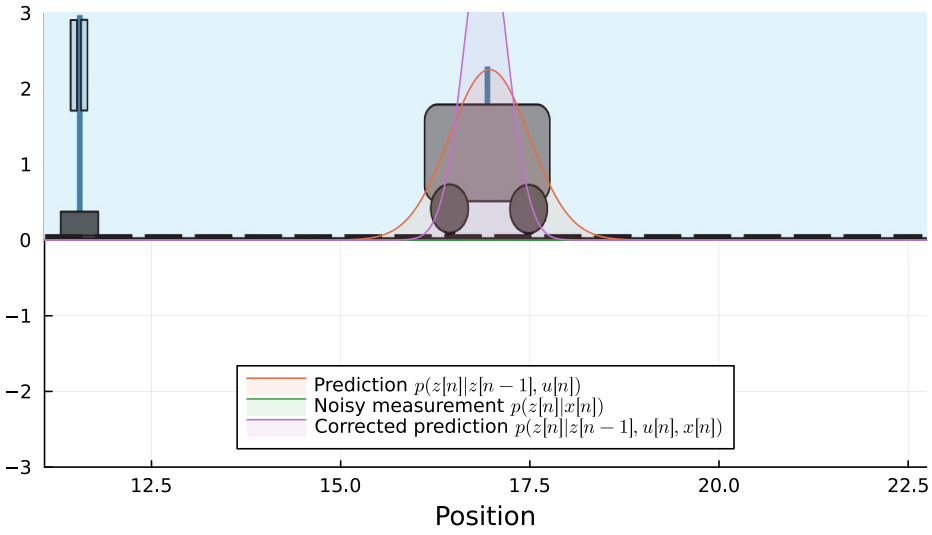
In [4]:
```
z_prev_m_0 = noisy_x[1]
z_prev_v_0 = A * (1e8*diageye(2) * A') + Σz ;

result = infer(
    model=cart_tracking(n=n, A=A,B=b, Σz=Σz, Σx=Σx, z_prev_m_0=z_prev_m_0, z_prev_v_0=z_prev_v_0,u=u),
    data=(x=noisy_x,),
    free_energy=true
);
```

In [5]:
```
import RxInfer.ReactiveMP: messageout, getinterface, materialize!
```

```
import RxInfer.Rocket: getrecent

which_timestep = 3

if which_timestep == 1
    z_prev_m, z_prev_S = mean_cov(result.posteriors[:z_prior])
else
    z_prev_m, z_prev_S = mean_cov(result.posteriors[:z][which_timestep-1])
end
μz_prediction, Σz_prediction = (A*z_prev_m + b*u[which_timestep], A*z_prev_S*A' + Σz)
μz_posterior, Σz_posterior = mean_cov.(result.posteriors[:z])[which_timestep]

println("Prediction: ",MvNormalMeanCovariance(μz_prediction, Σz_prediction))
println("Measurement: ", MvNormalMeanCovariance(noisy_x[which_timestep], Σx))
println("Posterior: ", MvNormalMeanCovariance(μz_posterior, Σz_posterior))
plotCartPrediction(μz_prediction[1], Σz_prediction[1], μz_posterior[1], Σz_posterior[1], noisy_x[n][1], Σx[1
```

```
Prediction: MvNormalMeanCovariance(
μ: [16.966508824183766, 2.527389683202836]
Σ: [0.5308027610923123 0.05645622740726185; 0.05645622740726185 0.21894113821638947]
)

Measurement: MvNormalMeanCovariance(
μ: [17.26380748518389, 2.0373239002146244]
Σ: [1.0 0.0; 0.0 2.0]
)

Posterior: MvNormalMeanCovariance(
μ: [16.91486174968641, 2.5570169803722296]
Σ: [0.2891731424490338 -0.030001367953500776; -0.030001367953500776 0.09410734708436251]
)
```

Out[5]:



- Note that both the analytical Kalman filtering solution and the message passing solution lead to the same results. The advantage of message passing-based inference with RxInfer is that we did not need to derive any inference equations. RxInfer took care of all that.

Recap Dynamical Models

- Dynamical systems do not obey the sample-by-sample independence assumption, but still can be specified, and state and parameter estimation equations can be solved by similar tools as for static models.
- Two of the more famous and powerful models with latent states include the hidden Markov model (with discrete states) and the Linear Gaussian dynamical system (with continuous states).
- For the LGDS, the Kalman filter is a well-known recursive state estimation procedure. The Kalman filter can be derived through Bayesian update rules on Gaussian distributions.

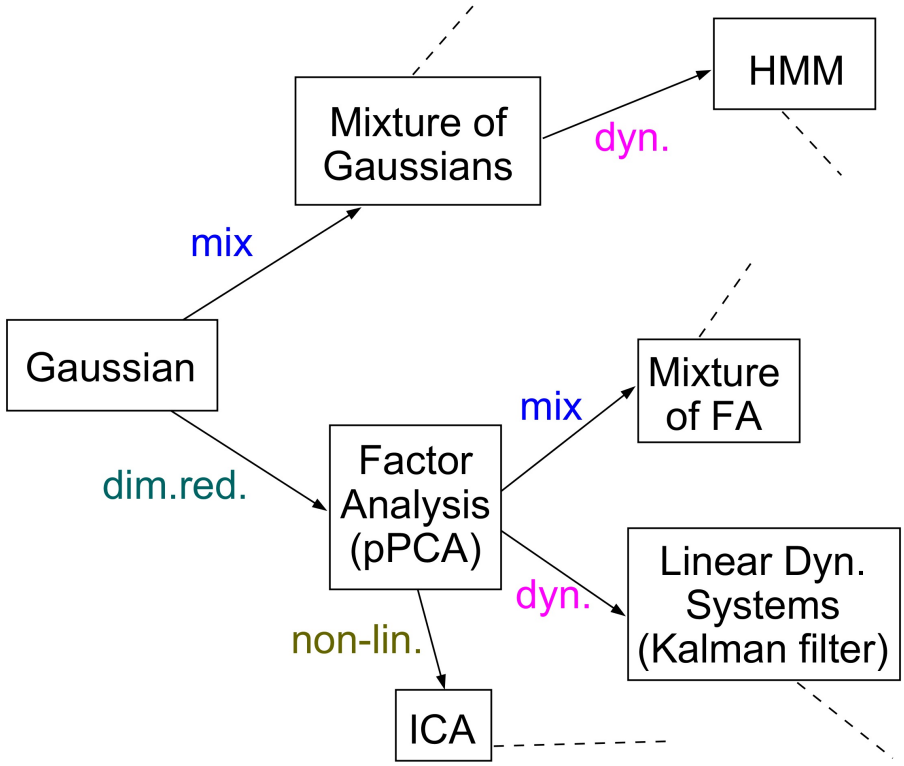
- If anything changes in the model, e.g., the state noise is not Gaussian, then you have to re-derive the inference equations again from scratch and it may not lead to an analytically pleasing answer.

- ⇒ Generally, we will want to automate the inference processes. As we discussed, message passing in a factor graph is a visually appealing method to automate inference processes. We showed how Kalman filtering emerged naturally by automated message passing.

OPTIONAL SLIDES

Extensions of Generative Gaussian Models

- Using the methods of the previous lessons, it is possible to create your own new models based on stacking Gaussian and categorical distributions in new ways:



Intelligent Agents and Active Inference

Preliminaries

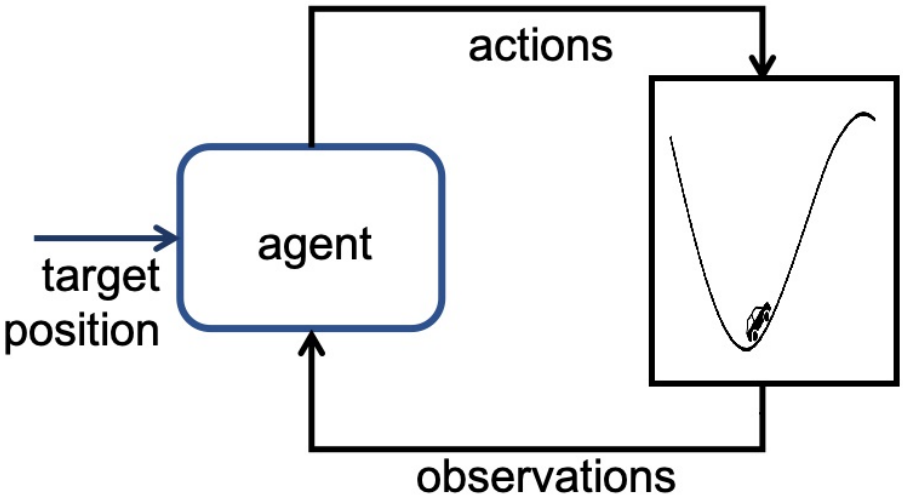
- Goal
  - Introduction to Active Inference and application to the design of synthetic intelligent agents
- Materials
  - Mandatory
    - These lecture notes
    - Bert de Vries - 2021 - Presentation on Beyond deep learning: natural AI systems (video)
  - Optional
    - Bert de Vries, Tim Scarfe and Keith Duggar - 2023 - Podcast on Active Inference. Machine Learning Street Talk podcast
      - Quite extensive discussion on many aspect regarding the Free Energy Principle and Active Inference, in particular relating to its implementation.
    - Raviv (2018), The Genius Neuroscientist Who Might Hold the Key to True AI.
      - Interesting article on Karl Friston, who is a leading theoretical neuroscientist working on a theory that relates life and intelligent behavior to physics (and Free Energy minimization). (**highly recommended**)
    - Friston et al. (2022), Designing Ecosystems of Intelligence from First Principles
      - Friston's vision on the future of AI.
    - Van de Laar and De Vries (2019), Simulating Active Inference Processes by Message Passing
      - How to implement active inference by message passing in a Forney-style factor graph.

Agents

- In the previous lessons we assumed that a data set was given.
- In this lesson we consider *agents*. An agent is a system that *interacts* with its environment through both sensors and actuators.
- Crucially, by acting onto the environment, the agent is able to affect the data that it will sense in the future.
  - As an example, by changing the direction where I look, I can affect the (visual) data that will be sensed by my retina.
- With this definition of an agent, (biological) organisms are agents, and so are robots, self-driving cars, etc.
- In an engineering context, we are particularly interesting in agents that behave with a *purpose* (with a goal in mind), e.g., to drive a car or to design a speech recognition algorithm.
- In this lesson, we will describe how **goal-directed behavior** by biological (and synthetic) agents can also be interpreted as minimization of a free energy functional.

Illustrative Example: the Mountain Car Problem

- In this example, we consider the mountain car problem which is a classical benchmark problem in the reinforcement learning literature.
- The car aims to drive up a steep hill and park at a designated location. However, its engine is too weak to climb the hill directly. Therefore, a successful agent should first climb a neighboring hill, and subsequently use its momentum to overcome the steep incline towards the goal position.
- We will assume that the agent's knowledge about the car's process dynamics (i.e., its equations of motion) are known up to some additive Gaussian noise.
- Your challenge is to design an agent that guides the car to the goal position. (The agent should be specified as a probabilistic model and the control signal should be formulated as a Bayesian inference task).



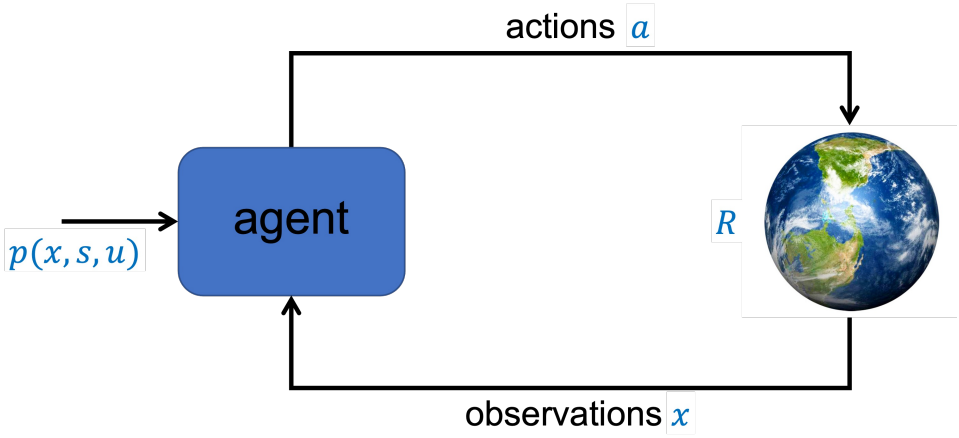
- Solution at the end of this lesson.

Karl Friston and the Free Energy Principle

- We begin with a motivating example that requires "intelligent" goal-directed decision making: assume that you are an owl and that you're hungry. What are you going to do?
- Have a look at Prof. Karl Friston's answer in this video segment on the cost function for intelligent behavior. (**Do watch the video!**)
- Friston argues that intelligent decision making (behavior, action making) by an agent requires *minimization of a functional of beliefs*.
- Friston further argues (later in the lecture and his papers) that this functional is a (variational) free energy (to be defined below), thus linking decision-making and acting to Bayesian inference.
- In fact, Friston's **Free Energy Principle** (FEP) claims that all biological self-organizing processes (including brain processes) can be described as Free Energy minimization in a probabilistic model.
  - This includes perception, learning, attention mechanisms, recall, acting and decision making, etc.
- Taking inspiration from FEP, if we want to develop synthetic "intelligent" agents, we have (only) two issues to consider:
  1. Specification of the FE functional.
  2. *How* to minimize the FE functional (often in real-time under situated conditions).
- Agents that follow the FEP are said to be involved in **Active Inference** (AIF). An AIF agent updates its states and parameters (and ultimately its model structure) solely by FE minimization, and selects its actions through (expected) FE minimization (to be explained below).

Execution of an AIF Agent

- Consider an AIF agent with observations (sensory states) x_t, latent internal states s_t and latent control states u_t for $t = 1, 2, \ldots$.



- The agent is embedded in an environment with "external states" $\tilde{s}_t$. The dynamics of the environment are driven by actions.
- Actions a_t are selected by the agent. Actions affect the environment and consequently affect future observations.
- In pseudo-code, an AIF agent executes the following algorithm:

  ACTIVE INFERENCE (AIF) AGENT ALGORITHM

  SPECIFY generative model $p(x, s, u)$
  ASSUME/SPECIFY environmental process R

  FORALL t DO

  1. $(x_t, \tilde{s}_t) = R(a_t, \tilde{s}_{t-1})$ % environment generates new observation
  2. $q(s_t) = \arg\min_q F[q]$ % update agent's internal states ("perception")
  3. $q(u_{t+1}) = \arg\min_q F_>[q]$ % update agent's control states ("actions")
  4. $a_{t+1} \sim q(u_{t+1})$ % sample next action and push to environment

  END

- In the above algorithm, $F[q]$ and $F_>[q]$ are appropriately defined Free Energy functionals, to be discussed below. Next, we discuss these steps in more details.

The Generative Model in an AIF agent

- What should the agent's model $p(x, s, u)$ be modeling? This question was (already) answered by Conant and Ashby (1970) as the *good regulator theorem*: **every good regulator of a system must be a model of that system**. See the OPTIONAL SLIDE for more information.
- Conant and Ashley state: "The theorem has the interesting corollary that the living brain, so far as it is to be successful and efficient as a regulator for survival, **must** proceed, in learning, by the formation of a model (or models) of its environment."
- Indeed, perception in brains is clearly affected by predictions about sensory inputs by the brain's own generative model.


- In the above picture (The Gardener, by Giuseppe Arcimboldo, ca 1590), on the left you will likely see a bowl of vegetables, while the same picture upside down elicits with most people the perception of a gardener's face rather than an upside-down vegetable bowl.
- The reason is that the brain's model predicts to see straight-up faces with much higher probability than upside-down vegetable bowls.
- So the agent's model p will be a model that aims to explain how environmental causes (latent states) lead to sensory observations.

Specification of AIF Agent's model and Environmental Dynamics

- In this notebook, for illustrative purposes, we specify the **generative model** at time step t of an AIF agent as

$$p(x_t, s_t, u_t|s_{t-1}) = \underbrace{p(x_t|s_t)}_{\text{observations}} \cdot \underbrace{p(s_t|s_{t-1}, u_t)}_{\substack{\text{state} \\ \text{transition}}} \cdot \underbrace{p(u_t)}_{\substack{\text{action} \\ \text{prior}}}$$

- We will assume that the agent interacts with an environment, which we represent by a dynamic model R as

$$(x_t, \tilde{s}_t) = R\left(a_t, \tilde{s}_{t-1}\right)$$

  where a_t are *actions* (by the agent), x_t are *outcomes* (the agent's observations) and $\tilde{s}_t$ holds the environmental latent *states*.
- Note that R only needs to be specified for simulated environments. If we were to deploy the agent in a real-world environment, we would not need to specify R.
- The agent's knowledge about environmental process R is expressed by its generative model $p(x_t, s_t, u_t|s_{t-1})$.
- Note that we distinguish between *control states* and *actions*. Control states u_t are latent variables in the agent's generative model. An action a_t is a realization of a control state as observed by the environment.
- Observations x_t are generated by the environment and observed by the agent. Vice versa, actions a_t are generated by the agent and observed by the environment.

State Updating in the AIF Agent

- After the agent makes a new observation x_t, it will update beliefs over its latent variables. First the internal state variables s.
- Assume the following at time step t:
  - the state of the agent's model has already been updated to $q(s_{t-1}|x_{1:t-1})$.
  - the agent has selected a new action a_t.
  - the agent has recorded a new observation x_t.
- The **state updating** task is to infer $q(s_t|x_{1:t})$, based on the previous estimate $q(s_{t-1}|x_{1:t-1})$, the new data $\{a_t, x_t\}$, and the agent's generative model.
- Technically, this is a Bayesian filtering task. In a real brain, this process is called **perception**.

- We specify the following FE functional

$$F[q] = \sum_{s_t} q(s_t|x_{1:t}) \log \frac{\overbrace{q(s_t|x_{1:t})}^{\text{state posterior}}}{\underbrace{p(x_t|s_t)p(s_t|s_{t-1},a_t)}_{\text{generative model w new data}} \underbrace{q(s_{t-1}|x_{1:t-1})}_{\text{state prior}}}$$

- The state updating task can be formulated as minimization of the above FE (see also AIF Algorithm, step 2):

$$q(s_t|x_{1:t}) = \arg\min_q F[q]$$

- In case the generative model is a *Linear Gaussian Dynamical System*, minimization of the FE can be solved analytically in closed-form and leads to the standard Kalman filter.
- In case these (linear Gaussian) conditions are not met, we can still minimize the FE by other means and arrive at some approximation of the Kalman filter, see for example Baltieri and Isomura (2021) for a Laplace approximation to variational Kalman filtering.
- Our toolbox RxInfer specializes in automated execution of this minimization task.

Policy Updating in an AIF Agent

- Once the agent has updated its internal states, it will turn to inferring the next action.
- In order to select a **good** next action, we need to investigate and compare consequences of a *sequence* of future actions.
- A sequence of future actions $a = (a_{t+1}, a_{t+2}, \dots, a_{t+T})$ is called a **policy**. Since relevant consequences are usually the result of an future action sequence rather than a single action, we will be interested in updating beliefs over policies.
- In order to assess the consequences of a selected policy, we will, as a function of that policy, run the generative model forward-in-time to make predictions about future observations $x_{t+1:t+T}$.
- Note that perception (state updating) preceeds policy updating. In order to accurately predict the future, the agent first needs to understand the current state of the world.
- Consider an AIF agent at time step t with (future) observations $x = (x_{t+1}, x_{t+2}, \dots, x_{t+T})$, latent future internal states $s = (s_t, s_{t+1}, \dots, s_{t+T})$, and latent future control variables $u = (u_{t+1}, u_{t+2}, \dots, u_{t+T})$.
- From the agent's viewpoint, the evolution of these future variables are constrained by its generative model, rolled out into the future:

$$p(x,s,u) = \underbrace{q(s_t)}_{\substack{\text{current} \\ \text{state}}} \cdot \underbrace{\prod_{k=t+1}^{t+T} p(x_k|s_k) \cdot p(s_k|s_{k-1},u_k) p(u_k)}_{\text{GM roll-out to future}}$$

- Consider the Free Energy functional for estimating posterior beliefs $q(s,u)$ over latent *future* states and latent *future* control signals:

$$\begin{aligned} F_>[q] &= \overbrace{\sum_{x,s} q(x|s)}^{\text{marginalize } x} \left(\overbrace{\sum_u q(s,u) \log \frac{q(s,u)}{p(x,s,u)}}^{\text{"regular" variational Free Energy}} \right) \\ &= \sum_{x,s,u} q(x,s,u) \log \frac{q(s,u)}{p(x,s,u)} \end{aligned}$$

- In principle, this is a regular FE functional, with one difference to previous versions: since future observations x have not yet occurred, $F_>[q]$ marginalizes not only over latent states s and policies u, but also over future observations x.
- We will update the beliefs over policies by minimization of Free Energy functional $F_>[q]$. In the optional slides below, we prove that the solution to this optimization task is given by (see AIF Algorithm, step 3, above)

$$\begin{aligned} q^*(u) &= \arg\min_q F_>[q] \\ &\propto p(u) \exp(-G(u)) \,, \end{aligned}$$

where the factor $p(u)$ is a prior over admissible policies, and the factor $\exp(-G(u))$ updates the prior with information about future consequences of a selected policy u.

- The function

$$G(u) = \sum_{x,s} q(x, s|u) \log \frac{q(s|u)}{p(x, s|u)}$$

is called the **Expected Free Energy** (EFE) for policy u.

- The FEP takes the following stance: if FE minimization is all that an agent does, then the only consistent and appropriate behavior for an agent is to select actions that minimize the **expected** Free Energy in the future (where expectation is taken over current beliefs about future observations).
- Note that, since $q^*(u) \propto p(u) \exp(-G(u))$, the probability $q^*(u)$ for selecting a policy u increases when EFE $G(u)$ gets smaller.
- Once the policy (control) variables have been updated, in simulated environments, it is common to assume that the next action a_{t+1} (an action is the *observed* control variable by the environment) gets selected in proportion to the probability of the related control variable (see AIF Agent Algorithm, step 4, above), i.e., the environment samples the action from the control posterior:

$$a_{t+1} \sim q(u_{t+1})$$

- Next, we analyze some properties of the EFE.

Active Inference Analysis: exploitation-exploration dilemma

- Consider the following decomposition of EFE:

$$\begin{aligned}
G(u) = \sum_{x,s} q(x, s|u) \log \frac{q(s|u)}{p(x, s|u)} \\
&= \sum_{x,s} q(x, s|u) \log \frac{1}{p(x)} + \sum_{x,s} q(x, s|u) \log \\
&\quad \frac{q(s|u)}{p(s|x, u)} \frac{q(s|x)}{q(s|x)} \\
&= \sum_{x} q(x|u) \log \frac{1}{p(x)} + \sum_{x,s} q(x, s|u) \log \\
&\quad \frac{q(s|u)}{q(s|x)} + \underbrace{\sum_{x,s} q(x, s|u) \log \frac{q(s|x)}{p(s|x, u)}}_{E[D_{\mathrm{KL}}[q(s|x), p(s|x,u)]] \geq 0} \\
&\geq \underbrace{\sum_{x} q(x|u) \log \frac{1}{p(x)}}_{\substack{\text{goal-seeking behavior} \\ \text{(exploitation)}}} \\
&\quad \underbrace{- \sum_{x,s} q(x, s|u) \log \frac{q(s|x)}{q(s|u)}}_{\substack{\text{information-seeking behavior} \\ \text{(exploration)}}}
\end{aligned}$$

- Apparently, minimization of EFE leads to selection of policies that balances the following two imperatives:

1. minimization of the first term of $G(u)$, i.e. minimizing $\sum_x q(x|u) \log \frac{1}{p(x)}$, leads to policies ($u$) that align the inferred observations $q(x|u)$ under policy u (i.e., predicted future observations under policy u) with a prior $p(x)$ on future observations. We are in control to choose any prior $p(x)$ and usually we choose a prior that aligns with desired (goal) observations. Hence, policies with low EFE leads to **goal-seeking behavior** (a.k.a. pragmatic behavior or exploitation). In the OPTIONAL SLIDES, we derive an alternative (perhaps clearer) expression to support this interpretation].
2. minimization of $G(u)$ maximizes the second term

$$\sum_{x,s} q(x,s|u) \log \frac{q(s|x)}{q(s|u)} = \sum_{x,s} q(x,s|u) \log \frac{q(s|x)}{q(s|u)} \frac{q(x|u)}{q(x|u)}$$
$$= \underbrace{\sum_{x,s} q(x,s|u) \log \frac{q(x,s|u)}{q(x|u)q(s|u)}}_{\text{(conditional) mutual information } I[x,s|u]}$$

which is the (conditional) **mutual information** between (posteriors on) future observations and states, for a given policy u. Thus, maximizing this term leads to actions that maximize statistical dependency between future observations and states. In other words, a policy with low EFE also leads to **information-seeking behavior** (a.k.a. epistemic behavior or exploration).

- (The third term $\sum_{x,s} q(x,s|u) \log \frac{q(s|x)}{p(s|x)}$ is an (expected) KL divergence between posterior and prior on the states. This can be interpreted as a complexity/regularization term and $G(u)$ minimization will drive this term to zero.)
- Seeking actions that balance goal-seeking behavior (exploitation) and information-seeking behavior (exploration) is a fundamental problem in the Reinforcement Learning literature.
- **Active Inference solves the exploration-exploitation dilemma**. Both objectives are served by EFE minimization without any need for tuning parameters.

AIF Agents learn both the Problem and Solution

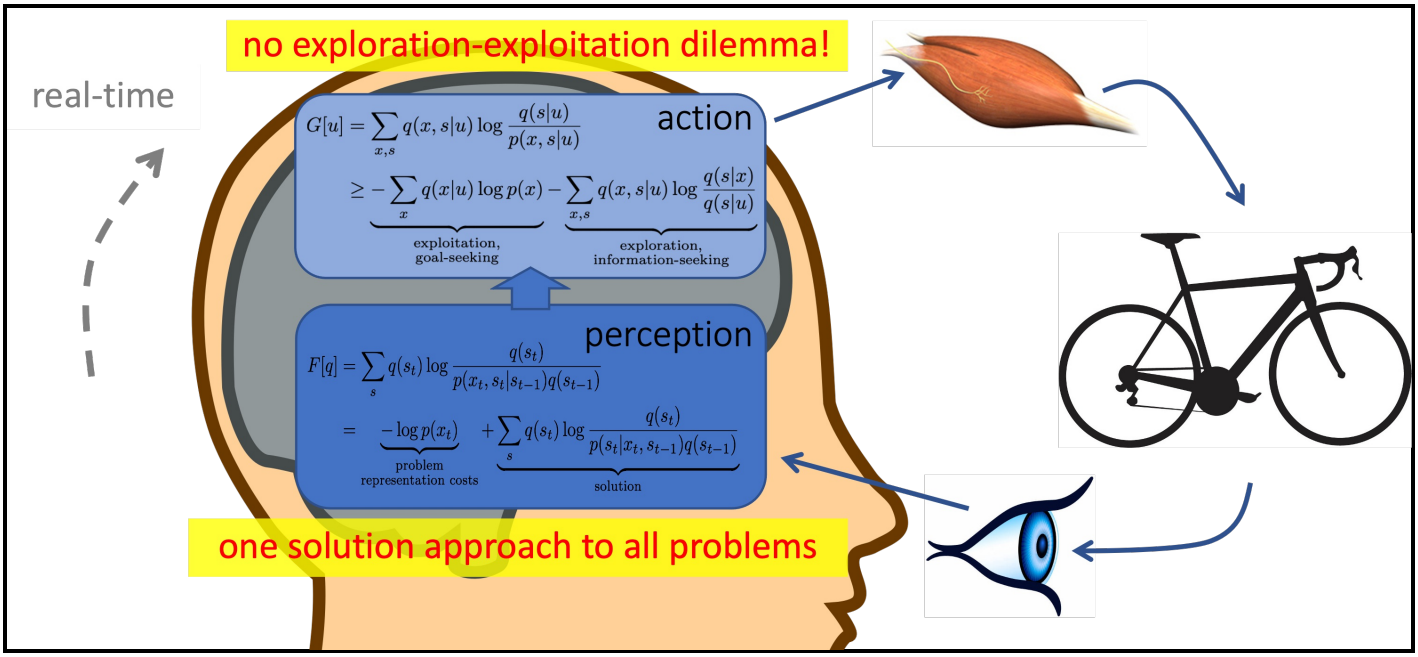
- We highlight another great feature of FE minimizing agents. Consider an AIF agent (m) with generative model $p(x,s,u|m)$.
- Consider the Divergence-Evidence decomposition of the FE again:

$$F[q] = \sum_{s,u} q(s,u) \log \frac{q(s,u)}{p(x,s,u|m)}$$
$$= \underbrace{-\log p(x|m)}_{\substack{\text{problem} \\ \text{representation costs}}} + \underbrace{\sum_{s,u} q(s,u) \log \frac{q(s,u)}{p(s,u|x,m)}}_{\text{solution costs}}$$

- The first term, $-\log p(x|m)$, is the (negative log-) evidence for model m, given recorded data x.
- Minimization of FE maximizes the evidence for the given model. The model captures the **problem representation**. A model with high evidence predicts the data well and therefore "understands the world".
- The second term scores the cost of inference. In almost all cases, the solution to a problem can be phrased as an inference task on the generative model. Hence, the second term **scores the accuracy of the inferred solution**, for the given model.
- FE minimization optimizes a balanced trade-off between a good-enough problem representation and a good-enough solution proposal for that model. Since FE comprises both a cost for solution *and* problem representation, it is a neutral criterion that applies across a very wide set of problems.
- A good solution to the wrong problem is not good enough. A poor solution to a great problem statement is not sufficient either. In order to solve a problem well, we need both to represent the problem correctly (high model evidence) and we need to solve it well (low inference costs).

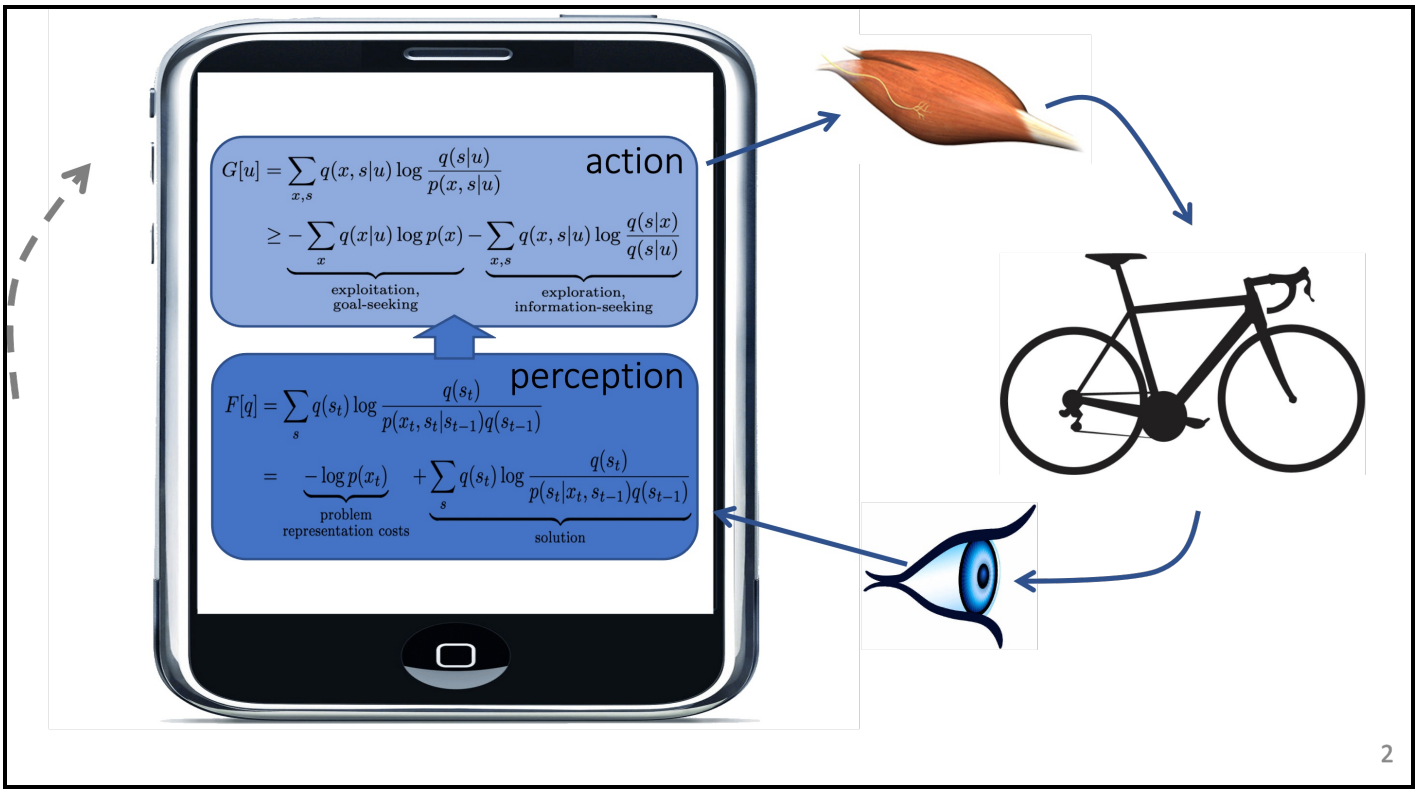
The Brain's Action-Perception Loop by FE Minimization

- The above derivations are not trivial, but we have just shown that FE-minimizing agents accomplish variational Bayesian perception (a la Kalman filtering), and a balanced exploration-exploitation trade-off for policy selection.
- Moreover, the FE by itself serves as a proper objective across a very wide range of problems, since it scores both the cost of the problem statement and the cost of inferring the solution.
- The current FEP theory claims that minimization of FE (and EFE) is all that brains do, i.e., FE minimization leads to perception, policy selection, learning, structure adaptation, attention, learning of problems and solutions, etc.



The Engineering Challenge: Synthetic AIF Agents

- We have here a framework (the FEP) for emergent intelligent behavior in self-organizing biological systems that
  - leads to optimal (Bayesian) information processing, including balancing accuracy vs complexity.
  - leads to balanced and continual learning of both problem representation and solution proposal
  - actively selects data in-the-field under situated conditions (no dependency on large data base)
  - pursues a optimal trade-off between exploration (information-seeking) and exploitation (goal-seeking) behavior
  - needs no external tuning parameters (such as step sizes, thresholds, etc.)
- Clearly, the FEP, and synthetic AIF agents as a realization of FEP, comprise a very attractive framework for all things relating to AI and AI agents.
- A current big AI challenge is to design synthetic AIF agents based solely on FE/EFE minimization.



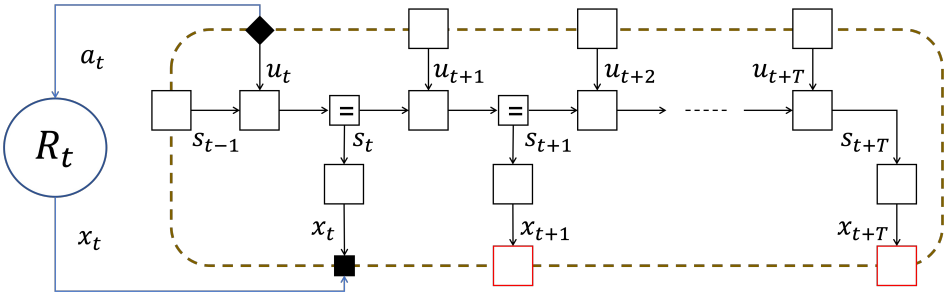
- Executing a synthetic AIF agent often poses a large computational problem because of the following reasons:
  1. For interesting problems (e.g. speech recognition, scene analysis), generative models may contain thousands of latent variables.

2. The FE function is a time-varying function, since it is also a function of observable variables.
3. An AIF agent must execute inference in real-time if it is engaged and embedded in a real world environment.

- So, in practice, executing a synthetic AIF agent may lead to a **task of minimizing a time-varying FE function of thousands of variables in real-time**!!

Factor Graph Approach to Modeling of an Active Inference Agent

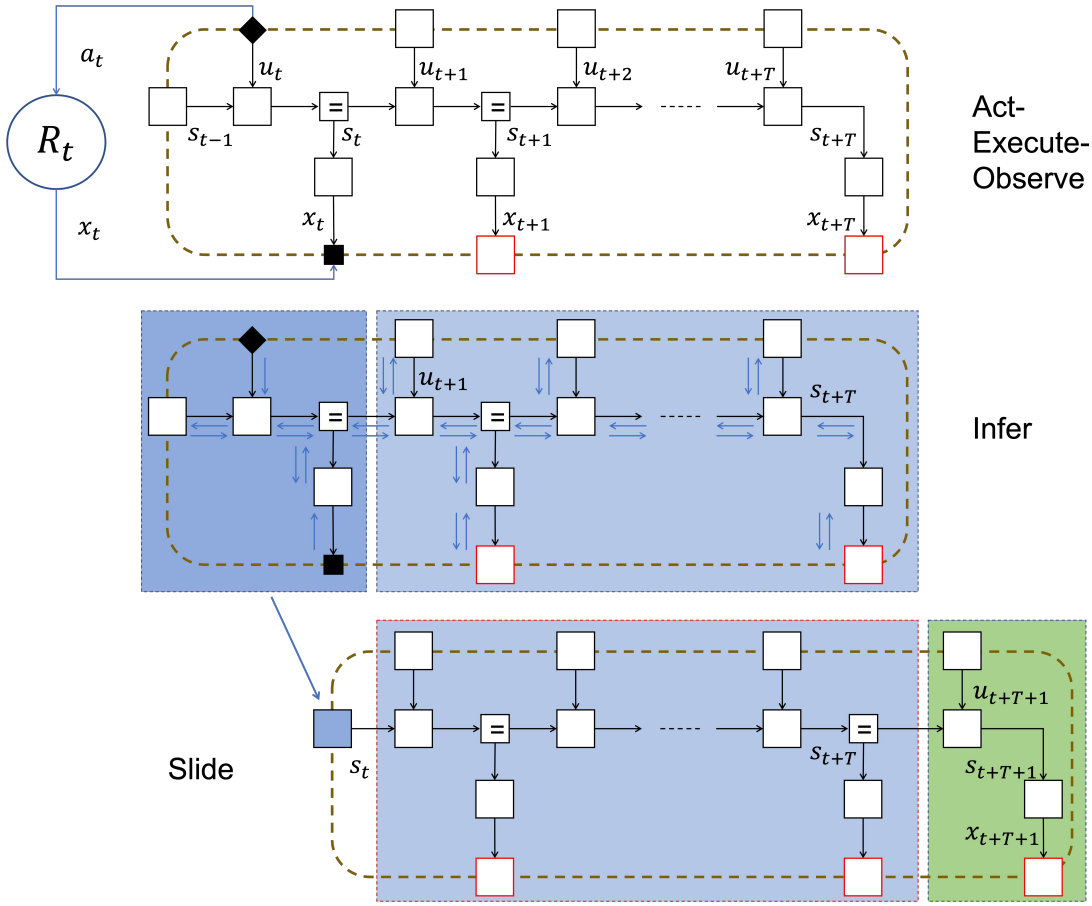
- How to specify and execute a synthetic AIF agent is an active area of research.
- There is no definitive solution approach to AIF agent modeling yet; we (BIASlab) think that (reactive) message passing in a factor graph representation provides a promising path.
- After selecting an action a_t and making an observation x_t, the FFG for the rolled-out generative model is given by the following FFG:



- The open red nodes for $p(x_{t+k})$ specify **desired future observations**, whereas the open black boxes for $p(s_k|s_{k-1}, u_k)$ and $p(x_k|s_k)$ reflect the agent's beliefs about how the world actually evolves (ie, the **veridical model**).
- The (brown) dashed box is the agent's Markov blanket. Given the states on the Markov blanket, the internal states of the agent are independent of the state of the world.

How to minimize FE: Online Active Inference

- Online active inference proceeds by iteratively executing three stages:
  1. act-execute-observe
  2. infer: update the latent variables and select an action
  3. slide forward



The Mountain car Problem Revisited

Here we solve the mountain car problem as stated at the beginning of this lesson. Before implementing the active inference agent, let's first perform a naive approach that executes the engine's maximum power to reach the goal. As can be seen in the results, this approach fails since the car's engine is not strong enough to reach the goal directly.

In [1]:
```
using Pkg; Pkg.activate("../."); Pkg.instantiate();
using IJulia; try IJulia.clear_output(); catch _ end
```

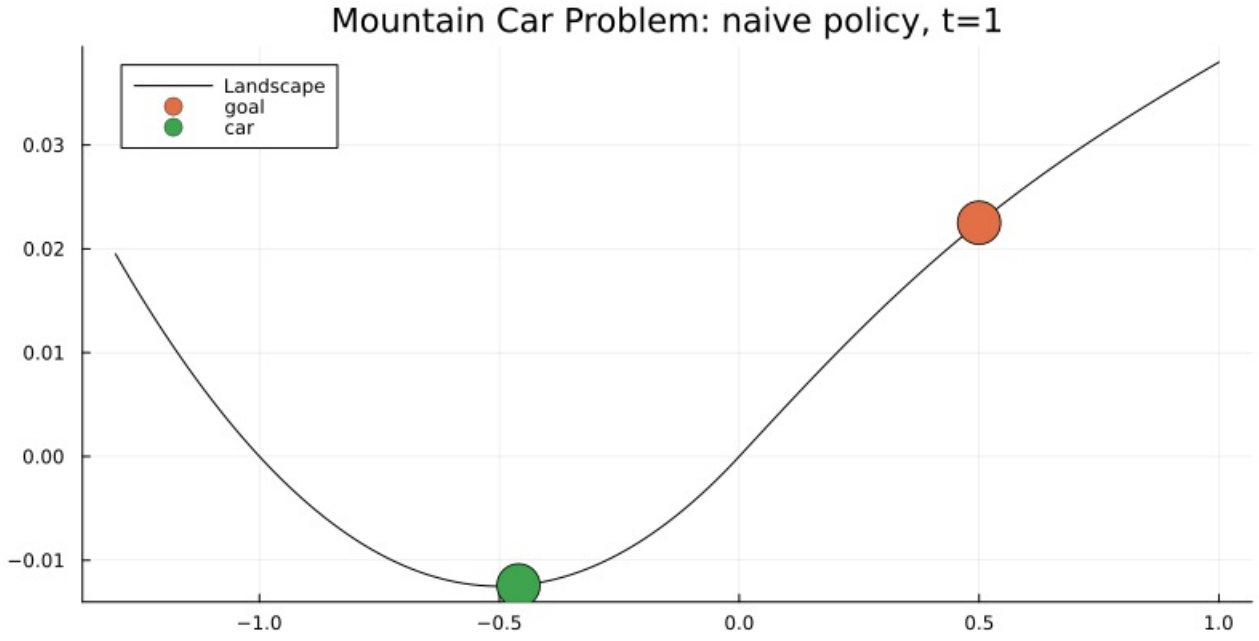
Out[1]:0

In [2]:
```
using LinearAlgebra, Plots, RxInfer
import .ReactiveMP: getrecent, messageout
include("./scripts/mountaincar_helper.jl")

# Environment variables
initial_position     = -0.5
initial_velocity     =  0.0
engine_force_limit   =  0.04
friction_coefficient =  0.1

Fa, Ff, Fg, height = create_physics(
    engine_force_limit = engine_force_limit,
    friction_coefficient = friction_coefficient
)

# Target position and velocity
target = [0.5, 0.0];
```

In [3]:
```
# Simulation of a naive policy, going full power toward the parking place

# Let there be a world
(execute, observe) = create_world(
    Fg = Fg, Ff = Ff, Fa = Fa,
    initial_position = initial_position,
    initial_velocity = initial_velocity
)

# Total simulation time
```

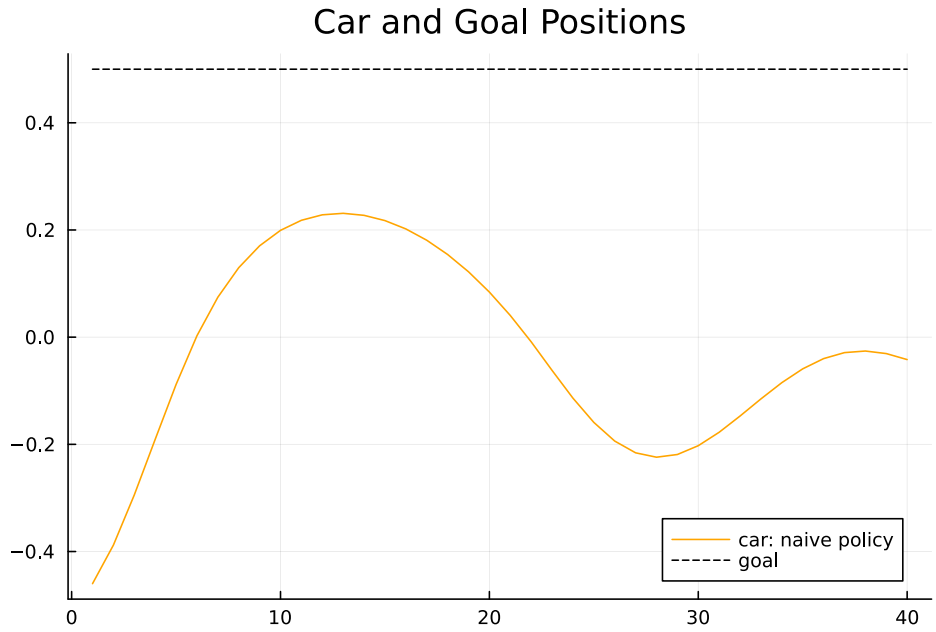
```
N = 40

y = Vector{Vector{Float64}}(undef, N)
for n in 1:N
    execute(100.0)       # Act with the maximum power
    y[n] = observe()     # Observe the current environmental outcome
end

plot_car(y, target, title_plot="Mountain Car Problem: naive policy", fps=5)
```

Out[3]:



In [4]:
```
# Let's also plot the goal and car positions over time
trajectories = reduce(hcat,y)'
plot(trajectories[:,1], label="car: naive policy", title = "Car and Goal Positions", color = "orange")
plot!(0.5 * ones(N), color = "black", linestyle=:dash, label = "goal")
```

Out[4]:



Next, we try a more sophisticated active inference agent. Above, we specified a probabilistic generative model for the agent's environment and then constrained future observations by a prior distribution that is located on the goal position. We then execute the (1) Act-execute-observe --> (2) infer --> (3) slide procedures as discussed above to infer future actions.

In [5]:
```
@model function mountain_car(m_u, V_u, m_x, V_x, m_s_t_min, V_s_t_min, T, Fg, Fa, Ff, engine_force_limit)

    # Transition function modeling transition due to gravity and friction
    g = (s_t_min::AbstractVector) -> begin

        s_t    = similar(s_t_min)                          # Next state
        s_t[2] = s_t_min[2] + Fg(s_t_min[1]) + Ff(s_t_min[2])   # Update velocity
```

```
        s_t[1] = s_t_min[1] + s_t[2]                             # Update position

        return s_t
    end

    # Function for modeling engine control
    h = (u::AbstractVector) -> [0.0, Fa(u[1])]

    # Inverse engine force, from change in state to corresponding engine force
    h_inv = (delta_s_dot::AbstractVector) -> [atanh(clamp(delta_s_dot[2], -engine_force_limit+1e-3, engine_f

    # Internal model perameters
    Gamma = 1e4*diageye(2)      # Transition precision
    Theta = 1e-4*diageye(2)     # Observation variance

    s_t_min ~ MvNormal(mean = m_s_t_min, cov = V_s_t_min)
    s_k_min = s_t_min

    local s

    for k in 1:T
        u[k]       ~ MvNormal(mean = m_u[k], cov = V_u[k])
        u_h_k[k]   ~ h(u[k]) where { meta = DeltaMeta(method = Linearization(), inverse = h_inv) }
        s_g_k[k]   ~ g(s_k_min) where { meta = DeltaMeta(method = Linearization()) }
        u_s_sum[k] ~ s_g_k[k] + u_h_k[k]
        s[k]       ~ MvNormal(mean = u_s_sum[k], precision = Gamma)
        x[k]       ~ MvNormal(mean = s[k], cov = Theta)
        x[k]       ~ MvNormal(mean = m_x[k], cov = V_x[k])      # goal

        s_k_min    = s[k]
    end

    return (s, )
end

@meta function car_meta()
    dzdt() -> DeltaMeta(method = Linearization())
end

function create_agent(;T = 20, Fg, Fa, Ff, engine_force_limit, target, initial_position, initial_velocity)

    Epsilon   = fill(huge, 1, 1)                        # Control prior variance
    m_u       = Vector{Float64}[ [ 0.0] for k=1:T ]     # Set control priors
    V_u       = Matrix{Float64}[ Epsilon for k=1:T ]

    Sigma     = 1e-4*diageye(2)                         # Goal prior variance
    m_x       = [zeros(2) for k=1:T]
    V_x       = [huge*diageye(2) for k=1:T]
    V_x[end]  = Sigma                                   # Set prior to reach goal at t=T

    # Set initial brain state prior
    m_s_t_min = [initial_position, initial_velocity]
    V_s_t_min = tiny * diageye(2)

    # Set current inference results
    result = nothing

    # The `compute` function performs Bayesian inference by message passing
    compute = (upsilon_t::Float64, y_hat_t::Vector{Float64}) -> begin

        m_u[1] = [ upsilon_t ]              # Register action with the generative model
        V_u[1] = fill(tiny, 1, 1)           # Clamp control prior to performed action

        m_x[1] = y_hat_t                    # Register observation with the generative model
        V_x[1] = tiny*diageye(2)            # Clamp goal prior to observation

        data = Dict(:m_u       => m_u,
                    :V_u       => V_u,
                    :m_x       => m_x,
                    :V_x       => V_x,
                    :m_s_t_min => m_s_t_min,
                    :V_s_t_min => V_s_t_min)

        model  = mountain_car(T=T, Fg=Fg, Fa=Fa, Ff=Ff, engine_force_limit=engine_force_limit)
        result = infer(model = model, data = data)
    end
```

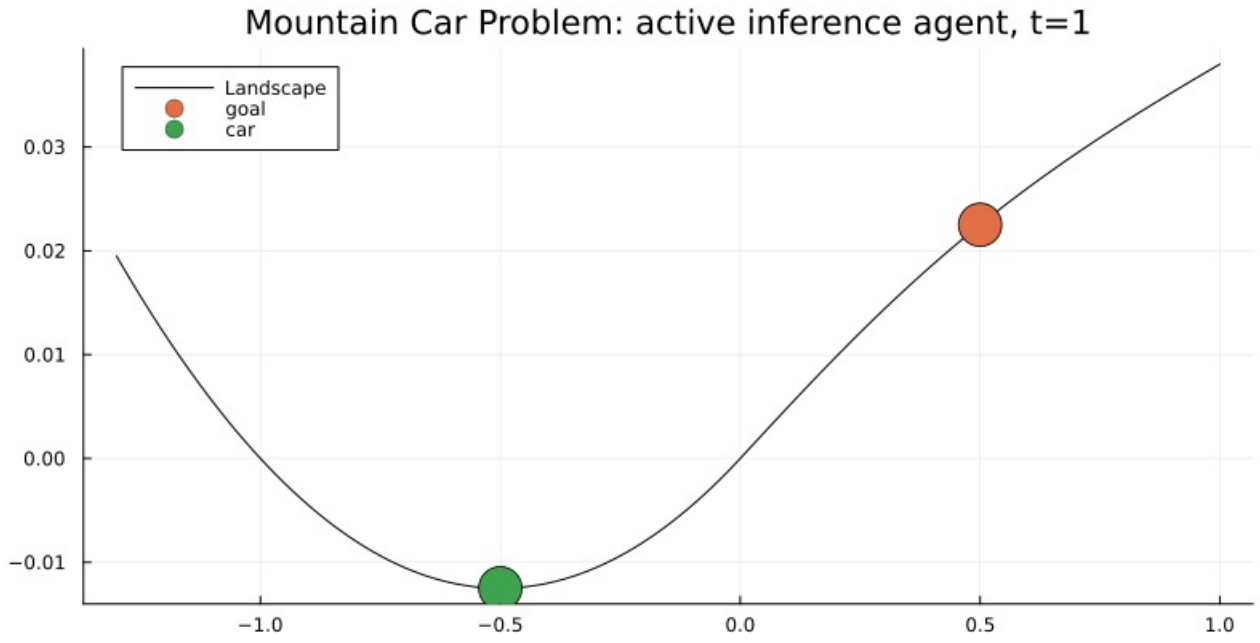
```
    # The `act` function returns the inferred best possible action
    act = () -> begin
        if result !== nothing
            return mode(result.posteriors[:u][2])[1]
        else
            # Without inference result we return some 'random' action
            return 0.0
        end
    end

    # The `future` function returns the inferred future states
    future = () -> begin
        if result !== nothing
            return getindex.(mode.(result.posteriors[:s]), 1)
        else
            return zeros(T)
        end
    end

    # The `slide` function modifies the `(m_s_t_min, V_s_t_min)` for the next step
    # and shifts (or slides) the array of future goals `(m_x, V_x)` and inferred actions `(m_u, V_u)`
    slide = () -> begin

        model  = RxInfer.getmodel(result.model)
        (s, )  = RxInfer.getreturnval(model)
        varref = RxInfer.getvarref(model, s)
        var    = RxInfer.getvariable(varref)

        slide_msg_idx = 3       # This index is model dependent
        (m_s_t_min, V_s_t_min) = mean_cov(getrecent(messageout(var[2], slide_msg_idx)))

        m_u      = circshift(m_u, -1)
        m_u[end] = [0.0]
        V_u      = circshift(V_u, -1)
        V_u[end] = Epsilon

        m_x      = circshift(m_x, -1)
        m_x[end] = target
        V_x      = circshift(V_x, -1)
        V_x[end] = Sigma
    end

    return (compute, act, slide, future)
end
```

Out[5]: create_agent (generic function with 1 method)

In [6]:
```
# Create another world
(execute_ai, observe_ai) = create_world(
    Fg = Fg,
    Ff = Ff,
    Fa = Fa,
    initial_position = initial_position,
    initial_velocity = initial_velocity
)

# Planning horizon
T_ai = 50

# Let there be an agent
(compute_ai, act_ai, slide_ai, future_ai) = create_agent(;

    T  = T_ai,
    Fa = Fa,
    Fg = Fg,
    Ff = Ff,
    engine_force_limit = engine_force_limit,
    target             = target,
    initial_position   = initial_position,
    initial_velocity   = initial_velocity
)

# Length of trial
N_ai = 100

# Step through experimental protocol
```

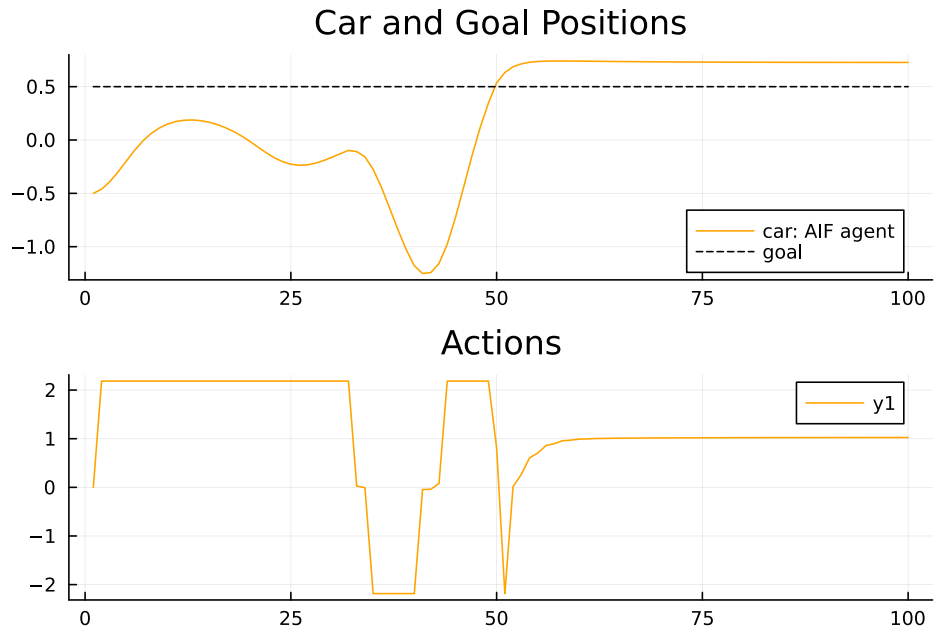
```
agent_a = Vector{Float64}(undef, N_ai)          # Actions
agent_f = Vector{Vector{Float64}}(undef, N_ai)  # Predicted future
agent_x = Vector{Vector{Float64}}(undef, N_ai)  # Observations

for t=1:N_ai
    agent_a[t] = act_ai()                # Invoke an action from the agent
    agent_f[t] = future_ai()             # Fetch the predicted future states
    execute_ai(agent_a[t])               # The action influences hidden external states
    agent_x[t] = observe_ai()            # Observe the current environmental outcome (update p)
    compute_ai(agent_a[t], agent_x[t])   # Infer beliefs from current model state (update q)
    slide_ai()                           # Prepare for next iteration
end

plot_car(agent_x, target, title_plot="Mountain Car Problem: active inference agent", fps=5)
```

Out[6]:



In [7]:
```
# Again, let's plot the goal and car positions over time
trajectories = reduce(hcat, agent_x)'
p1 = plot(trajectories[:,1], label="car: AIF agent", title = "Car and Goal Positions", color = "orange")
plot!(0.5 * ones(N_ai), color = "black", linestyle=:dash, label = "goal")
p2 = plot(agent_a, title = "Actions", color = "orange")
plot(p1,p2, layout = @layout [a ; b])
```

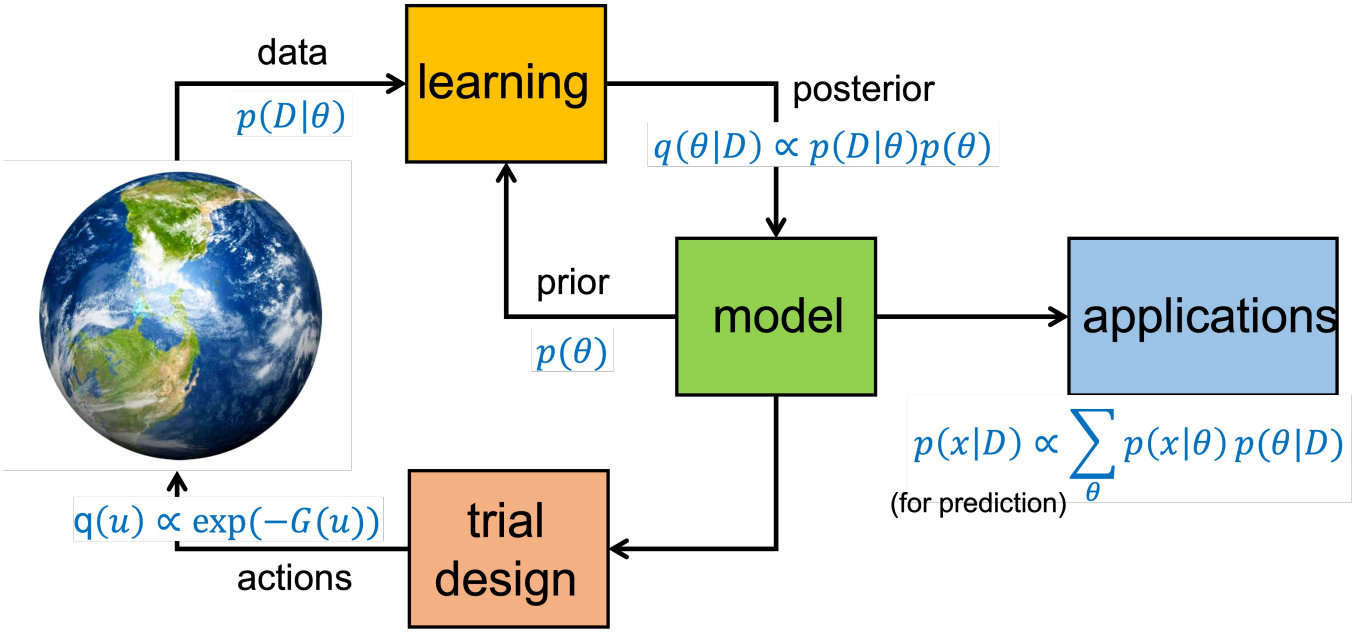
Out[7]:



Note that the AIF agent **explores** other options, like going first in the opposite direction of the goal prior, to reach its goals. This agent is able to mix exploration (information-seeking behavior) with exploitation (goal-seeking behavior).

Extensions and Comments

- Just to be sure, you don't need to memorize all FE/EFE decompositions nor are you expected to derive them on-the-spot. We present these decompositions only to provide insight into the multitude of forces that underlie FEM-based action selection.
- In a sense, the FEP is an umbrella for describing the mechanics and self-organization of intelligent behavior, in man and machines. Lots of sub-fields in AI, such as reinforcement learning, can be interpreted as a special case of active inference under the FEP, see e.g., Friston et al., 2009.
- Is EFE minimization really different from "regular" FE minimization? Not really, it appears that EFE minimization can be reformulated as a special case of FE minimization. In other words, FE minimization is still the only game in town.
- Active inference also completes the "scientific loop" picture. Under the FEP, experimental/trial design is driven by EFE minimization. Bayesian probability theory (and FEP) contains all the equations for running scientific inquiry.



- Essentially, AIF is an automated Scientific Inquiry Loop with an engineering twist. If there would be no goal prior, AIF would just lead to learning of a veridical ("true") generative model of the environment. This is what science is about. However, since we have goal prior constraints in the generative model, AIF leads to generating behavior (actions) with a purpose! For instance, when you want to cross a road, the goal prior "I am not going to get hit by a car", leads to inference of behavior that fulfills that prior. Similarly, through appropriate goal priors, the brain is able to design algorithms for object recognition, locomotion, speech generation, etc. In short, **AIF is an automated Bayes-optimal engineering design loop**!!
- The big engineering challenge remains the computational load of AIF. The human brain consumes about 20 Watt and the neocortex only about 4 Watt (which is about the power consumption of a bicycle light). This is multiple orders of magnitude (at least 1 million times) cheaper than what we can engineer on silicon for similar tasks.

Final Thoughts

- In the end, all the state inference, parameter estimation, etc., in this lecture series could have been implemented by FE minimization in an appropriately specified generative probabilistic model. However, the Free Energy Principle extends beyond state and parameter estimation. Driven by FE minimization, brains change their structure as well over time. In fact, the FEP extends beyond brains to a general theory for biological self-organization, e.g., Darwin's natural selection process may be interpreted as a FE minimization-driven model optimization process, and here's an article on FEP for predictive processing in plants. Moreover, Constrained-FE minimization (rephrased as the Principle of Maximum Relative Entropy) provides an elegant framework to derive most (if not all) physical laws, as Caticha exposes in his brilliant monograph on Entropic Physics. Indeed, the framework of FE minimization is known in the physics community as the very fundamental Principle of Least Action that governs the equations-of-motion in nature.
- So, the FEP is very fundamental and extends way beyond applications to machine learning. At our research lab at TU/e, we work on developing FEP-based intelligent agents that go out into the world and autonomously learn to accomplish a pre-determined task, such as learning-to-walk or learning-to-process-noisy-speech-signals. Free free to approach us if you want to know more about that effort.

OPTIONAL SLIDES

In an AIF Agent, Actions fulfill Desired Expectations about the Future

- In the derivations above, we decomposed the EFE into an upperbound on the sum of a goal-seeking and information-seeking term. Here, we derive an alternative (exact) decomposition that more clearly reveals the goal-seeking objective.
- We consider again the EFE and factorize the generative model $p(x, s|u) = p'(x)p(s|x, u)$ as a product of a **goal prior** $p'(x)$ on observations and a **veridical** state model $p(s|x, u)$.
- Through the **goal prior** $p'(x)$, the agent declares which observations it **wants** to observe in the future. (The prime is just to distinguish the semantics of a desired future from the model for the actual future).
- Through the **veridical** state model $p(s|x, u)$, the agent implicitly declares its beliefs about how the world will **actually** generate observations.
  - In particular, note that through the equality (by Bayes rule)

$$p(s|x, u) = \frac{p(x|s)p(s|u)}{p(x|u)} = \frac{p(x|s)p(s|u)}{\sum_s p(x|s)p(s|u)},$$

    it follows that in practice the agent may specify $p(s|x, u)$ implicitly by explicitly specifying a state transition model $p(s|u)$ and observation model $p(x|s)$.
- Hence, an AIF agent holds both a model for its beliefs about how the world will actually evolve AND a model for its beliefs about how it desires the world to evolve!!
- To highlight the role of these two models in the EFE, consider the following alternative EFE decomposition:

$$
\begin{aligned}
G(u) &= \sum_{x,s} q(x,s|u) \log \frac{q(s|u)}{p'(x)p(s|x,u)} \\
&= \sum_{x,s} q(x,s|u) \log \frac{q(s|u)}{p'(x)} \frac{1}{p(s|x,u)} \\
&= \sum_{x,s} q(x,s|u) \log \frac{q(s|u)}{p'(x)} \frac{p(x|u)}{p(x|s)p(s|u)} \\
&\qquad \text{(use Bayes)} \\
&= \sum_{x,s} q(x,s|u) \log \frac{q(s|u)}{p(x|s)p(s|u)} \frac{p(x|u)}{p'(x)} \\
&= \sum_{x,s} q(x,s|u) \log \frac{q(s|u)}{p(x|s)p(s|u)} + \\
&\qquad \sum_{x,s} q(x,s|u) \log \frac{p(x|u)}{p'(x)} \\
&= \sum_{x,s} p(s|u)p(x|s) \log \frac{p(s|u)}{p(x|s)p(s|u)} + \\
&\qquad \sum_{x,s} p(s|u)p(x|s) \log \frac{p(x|u)}{p'(x)} \\
&\qquad (\text{ assume } q(x,s|u) = p(x|s)p(s|u) \ ) \\
&= \sum_{s} p(s|u) \sum_{x} p(x|s) \log \frac{1}{p(x|s)} + \\
&\qquad \sum_{x} p(x|u) \log \frac{p(x|u)}{p'(x)} \\
&= \underbrace{E_{p(s|u)}\left[H[p(x|s)]\right]}_{\text{ambiguity}} + \underbrace{D_{\mathrm{KL}}\left[p(x|u), p'(x)\right]}_{\text{risk}}
\end{aligned}
$$

- In this derivation, we have assumed that we can use the generative model to make inferences in the "forward" direction. Hence, $q(s|u) = p(s|u)$ and $q(x|s) = p(x|s)$.
- The terms "ambiguity" and "risk" have their origin in utility theory for behavioral ecocomics. Minimization of EFE leads to minimizing both ambiguity and risk.
- Ambiguous (future) states are states that map to large uncertainties about (future) observations. We want to avoid those ambiguous states since it implies that the model is not capable to predict how the world evolves. Ambiguity can be resolved by selecting information-seeking (epistemic) actions.
- Minimization of the second term (risk) leads to choosing actions (u) that align **predicted** future observations (represented by $p(x|u)$) with **desired** future observations (represented by $p'(x)$). Agents minimize risk by selecting pragmatic (goal-seeking) actions.
- $\Rightarrow$ **Actions fulfill desired expectations about the future!**
- (return to related cell in main text).

Proof $q^*(u) = \arg\min_q F_>[q] \propto p(u)\exp(-G(u))$

- Consider the following decomposition:

$$
\begin{aligned}
F_{>}[q] &= \sum_{x,s,u} q(x,s,u) \log \frac{q(s,u)}{p(x,s,u)} \\
&= \sum_{x,s,u} q(x,s|u)q(u) \log \frac{q(s|u)q(u)}{p(x,s|u)p(u)} \\
&= \sum_{u} q(u) \Bigg(\sum_{x,s} q(x,s|u) \log \frac{q(s|u)q(u)}{p(x,s|u)p(u)} \Bigg) \\
&= \sum_{u} q(u) \Bigg(\log q(u) + \log \frac{1}{p(u)} + \underbrace{\sum_{x,s} q(x,s|u) \log \frac{q(s|u)}{p(x,s|u)}}_{G(u)} \Bigg) \\
&= \sum_{u} q(u) \log \frac{q(u)}{p(u)\exp(-G(u))}
\end{aligned}
$$

- This is a KL-divergence. Minimization of $F_{>}[q]$ leads to the following posterior for the policy:

$$
\begin{aligned}
q^*(u) &= \arg\min_q F_{>}[q] \\
&= \frac{1}{Z} p(u) \exp(-G(u))
\end{aligned}
$$

- (click to return to linked cell in the main text.)

What Makes a Good Agent? The Good Regulator Theorem

- According to Friston, an ``intelligent'' agent like a brain minimizes a variational free energy functional, which, in general, is a functional of a probability distribution p and a variational posterior q.
- What should the agent's model p be modeling? This question was (already) answered by Conant and Ashby (1970) as the Good Regulator Theorem: **every good regulator of a system must be a model of that system**.
- A Quote from Conant and Ashby's paper (this statement was later finessed by Friston (2013)):

> "The theory has the interesting corollary that the living brain, insofar as it is successful and efficient as a regulator for survival, *must* proceed, in learning, by the formation of a model (or models) of its environment."

Int. J. Systems Sci., 1970, vol. 1, No. 2, 89-97

EVERY GOOD REGULATOR OF A SYSTEM MUST BE A MODEL OF THAT SYSTEM[1]

Roger C. Conant

Department of Information Engineering, University of Illinois, Box 4348, Chicago, Illinois, 60680, U.S.A.

and W. Ross Ashby

Biological Computers Laboratory, University of Illinois, Urbana, Illinois 61801, U.S.A.[2]

[Received 3 June 1970]

The design of a complex regulator often includes the making of a model of the system to be regulated. The making of such a model has hitherto been regarded as optional, as merely one of many possible ways.

m this paper a theorem is presented which shows, under very broad conditions, that any regulator that is maximally both successful and simple *must* be isomorphic with the system being regulated. (The exact assumptions are given.) Making a model is thus necessary.

The theorem has the interesting corollary that the living brain, so far as it is to be successful and efficient as a regulator for survival, *must* proceed, in learning, by the formation of a model (or models) of its environment.

- (Return to related cell in main text).